A framework for modelling tropical forest dynamics

Allen C. Young

Thesis submitted in fulfilment of the requirements for
Doctor of Philosophy

University of Edinburgh
1999

Abstract

A framework for modelling the dynamics of tropical forests is described. The framework makes use of simulation models to predict the long term growth and yield of forests under different management regimes. It is designed to have practical application for the sustainable management of forest resources in tropical countries.

The framework comprises a suite of simulation models, each of which is appropriate in particular circumstances. Each model uses a disaggregated representation of a forest stand. An individual-based representation is used for trees in the forest stand above a threshold size, while a more aggregate representation (such as a cohort representation) is used for seedlings and saplings. Processes of stand disturbance and recovery from disturbance .are captured. Disturbance results from tree falls and from stand harvesting. Recovery from disturbance may involve seed production, seedling establishment, and competition between trees. Local interactions within the stand are captured, as is species-specific behaviour.

The content of each model in the framework is represented in a text-based model design. Details of the content are specified using a formal representation language. The language has semantics and syntax for specifying how a set of generic modelling concepts is employed in an individual model. Details specified in this way include the names and kinds of model variables, and the algorithms used in model calculations.

A formal-representation (an 'ontology') of the set of generic modelling concepts developed for use in the framework was created using the Ontolingua server. This provides an unambiguous specification of the modelling concepts used. This assists communication and may also make it easier to design rules and procedures for translating the model content into forms compatible with different modelling systems.

The framework has functionality to support the creation, implementation and documentation of new model designs. A standard text editor is used to create and edit new model designs. Designs are processed by tools in the framework to create programs and data sets necessary for use of the model. The Code Generator produces source-code appropriate for implementation of the model. The Source-code Compiler compiles the model source code to create a stand-alone model program. The Parameter Generator creates a parameter file suitable for use with the model and the Description Generator creates a natural language description of the model that is comprehensible, unambiguous and complete.

The framework supports the sharing and reuse of model content between models. Model content is packaged into modules that are stored in a special repository. Modules are stored in source-code rather than binary form and are incorporated in model source-code by the Code Generator program. Each module specifies an algorithm used in the model.

A common simulation interface is used with all models in the framework. This allows users to conduct simulation runs and vary model parameters. It also provides support for meaningful display of simulation data in graphs, tables and diagrams. Users of the framework can customise the data displays to a high-degree.

A case-study showing the application of the framework to an evaluation of alternative silvicultural regimes is reported. The investigation considered the impact of felling cycle length on growth and yield of timber for tropical forest in Kalimantan, Indonesia.

Acknowledgements

First and foremost I want to express my gratitude to my supervisor Robert Muetzelfeldt for his unflagging and irrepressible attention to detail, his support when things got tough, and his honesty. John Grace also helped - even if I did not see that much of him, his occasional 'pep-talks' over a glass of wine or two were a unique tuitionary experience.

Other colleagues in the Darwin Building played multifarious roles. In Room 117, Ted Wilson kept me right on silviculture and forestry, and Duncan Heathfield and Allan Kelly kept me right on computing. Kirsty Venner made me glad I did not have to count pine needles. Other friends in the Department who assisted over the years, if only by hearing my grief, include Eric Easton, Shiela Wilson, Mike Clearwater, Maree Lucas, Vicky Temperton, James Irvine Alistair Hamilton and Rob Clement.

I also have a debt of gratitude to all those Indonesian colleagues who forbore SYMFOR and contributed so many invaluable comments. Paian Sianturi, Prabianto Wibowo, Ai Sumanja, Ida Lanniari, Wisnu Rusamantoro, Anas Fauzi, Triyono Saputro, Ariya Hendrawan, Teguh Raharja all deserve a mention. Nearly all those I worked with remained calm, polite and friendly, occasionally (especially early on) in the face of diabolical behaviour by my software.

Thanks must also go to the ITFMP staff who made my trips to Indonesia run a lot more smoothly. Alistair Fraser, David Taylor, Budiawan, the ITFMP secratariat, Robert de Kock, Richard Scott, Simon Armstrong and Peter Wilkie all contributed.

Finally my thanks go to my family: my financial sponsors on the last leg of this journey, and as in everything else, my supporters throughout.

Glossary of abbreviations

AI	*Aggregate Information* - one or more items of information that is/are derived by an operation on one or more modelled entities *e.g.* the sum of tree volume for all trees in a plot.
AME	*Agroforestry Modelling Environment* - A diagram based modelling environment developed at the University of Edinburgh by Robert Muetzelfeldt and Jasper Taylor.
DDE	*Dynamic Data Exchange* - A protocol for exchange of data between processes for programs running under MS Windows.
DFD	*Data Flow Diagram* - A diagram showing the flow and transformation of data within a system.
DLL	*Dynamic Link Library* - A library of functions that is stored in binary form and that can be accessed by programs running under MS Windows.
GUI	*Graphical User Interface* - an interface in which users manipulate graphical symbols to interact with software.
I/O	*Input / Output* - relating to how a software component interacts with its environment.
IBM	*Individual-based model* - a model in which properties and behaviour of individuals of a population are represented.
JSD	*Jackson Structure Diagram* - a diagramming formalism suitable for representing the structure of data or processes.
KIF	*Knowledge Interchange Format* -a formal language based on first-order predicate logic.
MDI	*Multiple Document Interface* - an MS windows interface in which there is a single *parent* window and one or more *child* windows.
ME	*Modelled entity* - a crisply defined object that is used to capture part of the state of a modelled stand in SYMFOR simulations.
MS	*Microsoft corporation*
OOM	*Object-oriented modelling* - a form of modelling which emphasises the identification of crisply defined entities and the specification of their behaviour.
PSP	*Permanent sample plot* - a plot of vegetation which can be accurately located over a period of years so that repeat measurements can be obtained.
RAD	*Rapid application development* - a process of software development in which emphasis is placed on quickly obtaining a working system *e.g.* for purposes of soliciting feedback from stakeholders, rather than on *e.g.* optimising performance.
RDBMS	*Relational Database Management System* - a system for organising, storing and manipulating data.

SID	*Stand Initialisation Dataset* - a SYMFOR datastore that is used to specify the state of a modelled stand at the start of a simulation run.
SMC	*SYMFOR Model Compiler* - a SYMFOR application that is used to create new SYMFOR datastores using information contained in a Model Design
SMM	*SYMFOR Model Manager* - a SYMFOR application that is used to conduct simulation runs of SYMFOR model.
SSAD	*Structured System Analysis and Design* - a design methodology in which the emphasis is on
SYMFOR	*Sustainable Yield Models for tropical Forests* - a forest modelling system developed at the University of Edinburgh by Allen Young and Robert Muetzelfeldt.
TSD	*Time Series Dataset* - a SYMFOR datastore that is used to store the results produced by SYMFOR simulations.
VB	*Visual Basic* - a software development system produced by Microsoft that is based on the BASIC programming language.
VC	*Visual C* - a software development system produced by Microsoft that is based on the C programming language.

Table of contents

1. INTRODUCTION **1**

1.1 SUSTAINABLE EXPLOITATION OF TROPICAL FOREST 1

1.2 QUANTITATIVE MEASURES OF FOREST PRODUCTION 2

1.2.1 Role of production measures in sustainable forest management *2*

1.2.2 Variability in forest production *4*

1.2.3 Forest inventory for determination of forest production *5*

1.2.4 Modelling of forest production *6*

1.3 THE PROBLEM 8

1.4 CONTEXT OF THIS WORK 9

1.5 AIM AND OBJECTIVES 9

2. LITERATURE REVIEW **10**

2.1 TROPICAL FOREST DYNAMICS 11

2.1.1 Disturbance processes *11*

2.1.2 Recovery processes *12*

2.1.3 Important features of forest dynamics with respect to modelling *15*

2.2 MODELS OF FOREST DYNAMICS 16

2.2.1 Simple empirical models *18*

2.2.2 Size-class based models *18*

2.2.3 Cohort-based models *21*

2.2.4 Individual-based models *22*

2.2.5 Use of models in the context of sustainable forest management *25*

2.2.6 Common features of forest simulation models *26*

2.3 STRATEGIES AND SYSTEMS FOR DEVELOPMENT OF ECOLOGICAL MODELS 27

2.3.1 Stages in the model design process *27*

2.3.2 Handling changes to model design through time *31*

2.3.3 Sharing and reuse of modelling functionality *34*

2.3.4 Formulation of a strategy for model development *47*

3. REQUIREMENTS SPECIFICATION FOR SYMFOR 48

3.1 STAKEHOLDERS 48

3.1.1 Government Researchers 49

3.1.2 Research and Development personnel in Concession companies 50

3.1.3 University Researchers 50

3.1.4 Forestry Trainers 51

3.1.5 Forest Operations Managers 51

3.2 REQUIREMENTS 52

4. DESIGN OF SYMFOR ONTOLOGY 57

4.1 INTRODUCTION TO ONTOLOGIES 57

4.2 METHODS 60

4.2.1 Creation of informal ontology 60

4.2.2 Creation of the formal ontology 61

4.3 DESCRIPTION OF THE INFORMAL ONTOLOGY 64

4.3.1 Ontology diagrams 64

4.3.2 Overview of the SYMFOR Ontology 67

4.3.3 Definition of terms in the SYMFOR ontology 67

4.4 DISCUSSION 81

4.4.1 Comparison of SYMFOR ontology with other modelling ontologies 81

4.4.2 Use of Ontolingua for Ontology capture 83

4.4.3 Uses of ontologies 84

5. DESIGN OF SYMFOR SOFTWARE 86

5.1 SOFTWARE DESIGN - FUNCTIONAL VIEWPOINT 87

5.1.1 Terminators 91

5.1.2 Datastores 92

5.2.2 Processes 117

5.2 SOFTWARE DESIGN - DYNAMIC VIEWPOINT 130

5.3 DISCUSSION 134

6. IMPLEMENTATION OF SYMFOR SOFTWARE 136

6.1 SYMFOR MODEL COMPILER 138

6.1.1 Interface 138

6.1.2 Programming 139

6.2 SYMFOR MODEL MANAGER 142

6.2.1 Interface 146

6.2.2 Programming 174

6.3 SYMFOR DLL 170

6.4 DISCUSSION 173

7. SCENARIO ANALYSIS USING SYMFOR: GROWTH AND YIELD PROJECTION IN INDONESIAN FORESTS 175

7.1 INTRODUCTION TO INVESTIGATION 177

7.1.1 Aim 177

7.2 MODEL DESIGN 178

7.2.1 Trees 178

7.2.2 Gridsquare 185

7.2.3 Cohort 186

7.2.4 Fallen trees 187

7.2.5 Felled trees 187

7.2.6 Stand 188

7.2.7 Skidtrails 188

7.3 FORMAL MODEL DESIGN 192

7.4 PROCEDURE 193

7.5 SITE DESCRIPTION 194

7.6 RESULTS 195

7.7 DISCUSSION 202

7.7.1 Results from individual run 202

7.7.1.1 Volume production 202

7.7.2 Variability among replicate runs 204

7.7.3 Treatment and plot effects on MAH rate 205

7.7.4 Comparison with the design of other individual-based models 205

7.7.5 Possible improvements to model design 206

8. DISCUSSION **209**

8.1 FOREST PRODUCTION ESTIMATES 209
8.2 REPRESENTATION OF FOREST PROCESSES 211
8.3 DATA COLLECTION 214
8.4 FRAMEWORK ARCHITECTURE 216
8.5 MODELLING LANGUAGE CONCEPTS 218
8.6 USER INTERACTION 221
8.7 CONCLUDING REMARKS 223

REFERENCES **224**

APPENDIX 1: FORMAL REPRESENTATION OF SYMFOR ONTOLOGY **234**

APPENDIX 2: JOURNAL PAPER ON SYMFOR **240**

1. Introduction

1.1 SUSTAINABLE EXPLOITATION OF TROPICAL FORESTS

Tropical forests have been exploited by human societies for thousands of years. Forests of South-east Asia have yielded food, timber, fuel, medicines, spices, essential oils, gums, latexes, tannins, dyes, ornamental plants, wildlife (products and game), fodder and raw materials such as rattan, bamboo and fibre (Tamin, 1992). As well as products that are physically extracted, forests may provide many services such as carbon sequestration (Pinard and Putz, 1996) regulation of hydrological processes in catchments (Brunjizeel, 1993), and species conservation (Johns, 1997).

Management of tropical forests may be defined as intervention in the forest for the purpose of increasing its production. Most foresters are adamant that simply harvesting a good from the forest does not constitute management (*e.g.* Palmer and Synnot, 1992). Instead it may involve many different operations including replanting, thinning, cleaning and organisation of felling operations so as to promote future yields from the forest (*e.g.* Matthews, 1989).

Traditionally, sustainable management has been thought of as that form of management in which productive capacity does not decrease with time. The production referred to here is:

- on an area basis, *i.e.* excludes systems which maintain production by increasing area under management (Poore, 1989);
- refers to the physical yield of a single product such as timber.

Recently there have been moves to extend this conceptualisation to address certain problems (*e.g.* Maser, 1994; Salwasser *et al.*, 1993) and modifications to the concept of 'production' have been made. Two of these are that forest 'production' is now assumed to:

- encompass different products from the same forest area - an important part of maintaining forests and forest production may be to explicitly value services that were less acknowledged in the past;

- refer explicitly to products demanded by society - if a forest does not produce something demanded by society then it will be converted to another land use and will not persist. This can happen even if it is still giving the same level of output in terms of goods that were demanded by society in the past but are no longer required.

Under the new definition, rather than refinement of a single system of management, sustainability may involve adapting forest management to cope with two kinds of changes. The first kind is change in the capacity of an ecosystem to produce goods. For example, the nitrogen cycle has been substantially altered by anthropogenic activity, and there is accumulating evidence for climate change. These factors may change forest production in the future.

The second kind is change in societal demands. This may involve planning for a drop in demand for products currently required. For example, around the turn of the 20th century there was high demand for the species *Palaquim gutta* from the Malayan rainforest because its latex was suitable for use as the insulator in submarine telegraph cables. Since then substitutes have been found and demand has dropped substantially (Whitmore, 1988). It may also involve responding to new demands from society. For example, recently there has been much interest in the potential of forests to offset carbon emissions created by industrial applications (Pinard and Putz, 1996).

1.2 QUANTITATIVE MEASURES OF FOREST PRODUCTION

The development and use of quantitative measures of forest production has a long history in forestry (*e.g.* Assman, 1970; Davis and Johnson, 1987). In discussing this issue it is necessary to consider the reasons for developing such measures, features of the forest that are important in this context and the different methods that have been used in the past. Two features of the forest, which are important in this context, are its variation through space and its variation through time. Methods can be split into inventory (*i.e.* data collection) and modelling (*i.e.* the processing of sample data to arrive at estimates of forest production).

1.2.1 Role of production measures in sustainable forest management

It follows from the analysis given in Section 1.1 that sustainable forest management must include two kinds of activity that involve measures of forest production:

1. Evaluation of new production systems. Forest production estimates are made as part of a strategy to determine which production systems are ecologically and economically viable. Field trials that take the form of an experiment featuring one or more treatments and replication are often used. Permanent sample plots are used to facilitate recurrent measurement. Usually a large number of potential production systems are evaluated, though comparatively few are selected (*i.e.* the activity is undertaken on a speculative basis). Field trials are often performed in conjunction with an Environmental Impact Assessment.

2. Planning. Forest production estimates are used as a basis for taking management decisions such as the order in which management units are harvested. Planning uses tools such as yield tables and Geographical Information Systems.

It is important to realise that these uses of forest production estimates are not necessarily mutually exclusive. While many estimates found in the evaluation of new production systems will not be of use for planning purposes (as many of the production systems evaluated may never see the light of day), some systems will be adopted so that the estimates will be useful. Planning uses estimates that may be combined in yield tables - the planner does not need intimate knowledge of the origin of the estimates, just an indication of their accuracy, precision and generality.

It is also important to stress that evaluation of new production systems is an integral part of sustainable forest management. There has been a tendency to see this as at best a peripheral issue. For example, Poore (1989) argues that most of the technical problems of silviculture have been solved, and that those remaining for sustainable management of forests are organisational in nature. While it may be true that the extent of the organisational problems is greater than that of the technical problems, it is nevertheless true that evaluation of new production is critical to forest persistence for the reasons indicated in the last section.

1.2.2 Variability in forest production

For many goods or services the most important determinants of production are stand structure (*i.e.* the size class distribution of trees in the stand) and dynamics (*i.e.* change in stand structure through time). For example carbon sequestration potential, conservation of forest primates, and production of rattans may all depend on stand structure and dynamics (de Fretes, 1992; Pinard and Putz, 1996; Johns, 1997). In the case of timber production the link between forest structure and dynamics and production is especially close. This is because most silvicultural systems specify the extraction of larger trees in a stand such that forests with a stand structure in which there is high number of large trees give a higher timber yield. For example, in TPTI (the silvicultural system used in natural forests of Indonesia) all trees without stem defects (or other properties which make them not saleable) above 50 cm stem diameter are removed every 35 years (Anon, 1992). Forest dynamics, *i.e.* recovery of the stand after harvesting disturbance, are critical in determining second-cycle and subsequent yields. Both structure and dynamics of forest may vary from place to place.

Foresters and ecologists have recognised spatial variability in tropical forest structure for many years (Richards, 1952; Ashton, 1964; Whitmore, 1988). This variability may be related to latitude, elevation or soil type, presence of rivers, or seasonal dryness. For example, in Indonesian Borneo two kinds of forest occur in the lowlands: lowland dipterocarp and heath forest (Whitmore, 1988). They differ in both physionomy and in species composition, the heath forest generally being of much lower stature.

Processes of forest dynamics are also critical in determining forest production of some goods. These processes can be divided into two kinds: those associated with disturbance and those associated with recovery from disturbance. Disturbance can either be natural (not caused by the direct action of humans) or anthropogenic. Natural disturbance may be caused by many events such as fall of trees or branches within the forest, fire or drought. If a single disturbance event results in large changes to the forest then it is said to be 'catastrophic'. Fires or droughts are examples of such events. If a single disturbance event results in minor changes to the forest stand and occurs frequently then it is often labelled 'background' disturbance. Treefalls are examples of background disturbance events.

The nature of recovery after disturbance depends to some extent on the characteristics of the initiating disturbance. In particular the characteristics of gaps (*i.e.* interruptions in the forest canopy) created in the disturbance may be important (Brokaw, 1985). Different species infiltrate different gap sizes and may occur at different times of the succession. For example, in Indonesian Borneo light demanding species such as those of genus *Macaranga* typically occur in large gaps soon after the disturbance (Primack and Lee, 1991). Slower-growing species such as those of genus *Shorea* may not appear until gaps have closed over much later in the succession.

Anthropogenic disturbance is often associated with forest management. Management operations undertaken by industrialised exploitation agencies typically involve the use of heavy machinery within the stand. This gives rise to a number of phenomena not seen in natural disturbance such as removal of surface soil and soil compaction. For these reasons management disturbance results in more complex changes to a forest ecosystem than natural disturbance.

1.2.3 Forest inventory for determination of forest production

Any scheme for quantification of forest production must draw upon information collected in forest inventory. There are many different kinds of inventory and each may be suitable for different tasks (Philip, 1994). However, two main types can be recognised:

- Census. In this form of inventory a complete enumeration of the forest is performed. For example the number of harvestable trees in a forest can be determined by 'cruising' the forest prior to exploitation. This has the advantage that it gives a high degree of precision. It has the disadvantage that it only provides a snapshot of the forest: while it may give a good indication of first cycle yield, it does not give a good guide of second cycle yield. It is also expensive, as it involves 100 percent coverage of the forest.

- Sampling. In this form of inventory a partial enumeration of the forest is performed. This is the form of inventory usually practised, primarily because it is not as expensive (as it does not require 100 percent coverage of the forest area).

The latter form of inventory always involves some kind of modelling, as it requires extrapolation and/or interpolation of sample data to give figures that can be used in management. The production figures produced by using a combination of inventory based

on sampling and modelling are *estimates i.e.* figures that are used in lieu of actual data and to which a level of confidence can be attached.

1.2.4 Modelling of forest production

The nature of the sampling and modelling scheme used to provide production estimates will determine the properties of the estimates. Each estimate has three properties which determine the use it can be put to: precision, accuracy and generality (Vanclay, 1995).

In schemes utilising simple empirical models the production from one or more sample plots is assumed to hold good across a certain area of forest such as a management unit. These models have the advantage of being precise and accurate but are not very general *i.e.* the model predictions can only be used for the forest structure and management scenario that applied in the original plots. In addition results from sample plots only become available after a length of time deemed sufficient to establish the character of the production system. Usually this will be a time corresponding to 2 or more cycles of the system and is of the order of 50 - 100 years.

Simulation models are a class of model that provide greater generality while equalling other models in precision and accuracy. A number of different kinds of simulation model have been used for predicting long-term forest dynamics. Each of them uses a representation of the forest that is modified in a series of iterations to capture the development of the forest through time. (A more formal definition of simulation models can be found in Bossel, 1994.) The unit of representation that is used to capture the trees of the forest can be used to distinguish forest simulation models. Representation units that have been used in this class of model include tree size-classes, tree cohorts and individual trees.

Inclusion of ecological processes such as competition for light or other resources means that these models are able to capture more of the behaviour exhibited by real forests. For example, the inclusion of competition processes may mean that a model is more accurately able to capture recovery from disturbance. This is because competition may structure development of trees but will be variable in its intensity throughout a plot.

Individual-based forest models (*i.e.* models in which the unit of representation is an individual tree) have some properties that make them especially useful for assessing forest dynamics in response to management operations. In particular they avoid aggregation error which may occur with other modelling approaches (Huston, 1994; O'Neill and Rust, 1979), and they are good at capturing local (as opposed to stand-level) interactions (Judson, 1994).

The properties of simulation models make them particularly suitable for conducting evaluations of alternative production systems. As mentioned before they may achieve high generality by incorporating ecological processes. This means that they can be used to investigate many different production systems with little recalibration. There have been many attempts to use ecological simulation models for this purpose (*e.g.* Bossel and Krieger, 1991, 1994; Korsgarrd, 1989; Mendoza and Setayarso, 1986; Kurpick, Kurpick, and Huth, 1997). Most of these attempts share a number of features. First, they often involve the development of a single model to answer a specific well-defined question. Second, the model is implemented as a computer program by a human computer programmer. Third, the model implementation may be targeted at users with a high degree of modelling expertise. It may therefore not make strong use of techniques such as visualisation of simulation data that would make the modelling more accessible to users with less modelling expertise. Fourth, the model and results from using the model are reported in papers or other documents - the number of people with access to the model source code is limited. Fifth, adaptation of the model to deal with a new forest type, new ecological insights or to simulate a new management activity is a specialist activity that can only performed by the original programmer or someone with sufficient time to gain an intimate knowledge of the model and program.

While the above may be a valid and appropriate strategy for forest modelling in some circumstances, particularly for initial work, it is unlikely to be so in all circumstances. This is because the nature of modelling and the agents involved will inevitably be different if it is adopted by an organisation such as a commercial enterprise. For example, with the more widespread use of modelling that will attend its commercialisation, duplication of effort becomes more likely. If modellers are to work co-operatively then efforts must be made to maintain compatibility between their work.

Currently advances are being made in several fields that have potential to contribute to the development of a modelling system. In particular there are now many design methodologies which may be used to structure software development (Budgen, 1994). These offer several advantages. They are formal techniques so that they lend themselves to documentation. Many problems in design will have been encountered before in other domains so that the development methodology itself may suggest some solutions. In addition many development methodologies stress the importance of an initial phase for determining requirements for the software based on the characteristics of stakeholders (*i.e.* the people with a strategic interest in the software). This means that the software is more likely to be focussed on the stakeholders and so is more likely to be used.

1.3 THE PROBLEM

The analysis presented above suggests that there are currently several problems with respect to assessing forest productivity over time. First, at the highest level, there is the problem of evaluating large numbers of alternative systems (which is required for sustainable forest management) with limited resources. This problem is compounded by the spatial variability found in tropical forest and changes to the structure and dynamics of the system resulting from management activities. Second, at a lower level, there is the problem of using simulation models to address the first problem. Simulation models that exist often require a high level of technical expertise for their use and maintenance. Support for visualisation is often weak, and it is difficult to adapt existing models to address new problems resulting from changed circumstances.

1.4 CONTEXT OF THIS WORK

The work described in this thesis was undertaken as part of a bilateral international development programme. The UK-Indonesia Tropical Forest Management Programme was funded by the Overseas Development Administration (now Department for International Development) of the United Kingdom and the Ministry of Forestry of the Republic of Indonesia. The programme consisted of five projects that addressed issues relating to government policy and senior management, provincial level forest management, forest research, training for forestry professionals and forest conservation. The overall goal of the programme was to contribute to the wise utilisation of forest resources within Indonesia. The research component was charged with the development of a system for estimating second cycle yield for different forest types under different systems of forest management. The system developed was given the name SYMFOR (Sustainable Yield Models for Tropical Forests) and it is the design and implementation of SYMFOR that forms the basis of this thesis.

1.5 AIM AND OBJECTIVES

The aim of this work is to design and implement a framework that supports the design and use of simulation models for evaluating the productivity of different tropical forest management systems. In order to satisfy this aim the specific objectives of the work were:

I. *to review relevant literature on forest modelling and forest dynamics;*

II. *to undertake an analysis of requirements for a framework;*

III. *to undertake design of a framework capable of meeting the requirements;*

IV. *to implement the design as computer software;*

V. *to conduct a modelling case study using the framework;*

VI. *to evaluate the success of the work;*

2. Literature review

The overall purpose of this work is to produce a modelling tool that can be used to produce models that will inform and improve the management of tropical forests. The material relevant to this aim can be divided into three sections:

1. The characteristics of forest dynamics in managed forests. The categories of models considered in this thesis rely for their success on their ability to capture the *detail* of processes within forest stands in order to make predictions concerning their long-term behaviour. To evaluate such models it is necessary to not only consider how predictions of stand level statistics agree with empirical data but also to consider if the representation of mechanisms embodied within the model is consistent with our current understanding of reality (Oderwald and Hans, 1993).

2. The types of models that have been used to capture forest dynamics. Many different kinds of models have been used over the years for this purpose. Each kind has particular advantages and disadvantages, and not all are suitable for every application. In addition, each has certain features that determine how easy it is to implement in modelling systems. Therefore there is a need to identify suitable models and to identify the features of each kind that will determine if and how they can be implemented in modelling systems.

3. Modelling strategies and systems that have been used to develop models. Model development is seldom a well-ordered linear process. Models are frequently adapted to deal with new circumstances that arise over time. Different systems and strategies vary in their capacity to support this continual adaptation. Any system that is designed to be used widely or used for a reasonable length of time in the future must be able to handle changes in models. As the former is a requirement for this piece of work, there is therefore a need to review current systems and strategies and to evaluate their ability to handle change to models.

2.1 TROPICAL FOREST DYNAMICS

Dynamic processes can be divided into two kinds: disturbance processes and recovery processes. This division contrasts with that of splitting dynamic processes into those related to regeneration, growth and mortality (*e.g.* Alder, 1995; Vanclay 1995). However, disturbance is now recognised as a useful concept for describing and explaining dynamics in a wide range of ecosystems (*e.g.* Pickett and White 1985; Huston, 1994). In the case of forest ecosystems (as is discussed below) gross tree mortality rate is less important than horizontal distribution and character of mortality (*i.e.* the local disturbance created by mortality) in determining the nature of regrowth that takes place within the stand. It is possible for two forest stands with the same mortality rates to show substantially different patterns of regeneration, if mortality is clustered in one and dispersed in the other. It may therefore be more important for models of forest dynamics to capture disturbance than mortality *per se*.

2.1.1 Disturbance processes

Disturbance in forest stands originates in many ways including landslides (Garwood *et al.*, 1979), logging and drought (Woods, 1989), fire (Goldammer and Siebert, 1990) and treefalls and branch falls (Brokaw, 1985; Van der Meer and Bongers, 1996). Disturbances can be divided into two kinds: *background* and *catastrophic* (Lugo and Scatena, 1996). Background disturbance occurs continuously, and individual disturbances of this kind are limited in horizontal scale such that the affect on a stand of a single event is small. Treefalls and branchfalls within a stand are in this group. Catastrophic disturbance occurs irregularly and its horizontal extent is extended such that its affect on a stand is large. Loggings and drought fall into this category.

Background disturbance may be caused in a variety of ways. Large trees may die while still standing and lose their foliage. Branches from large trees may snap and damage other trees in the locality. The trunks of trees may snap or trees may be completely uprooted in storms. Secondary treefall, resulting from physical damage caused by impact of falling branches or trees may serve to concentrate canopy disruption in localised areas. Secondary treefall is particularly likely in forests in which trees are linked by lianas (Putz, 1984).

Openings in the canopy created by fall of trees or branches ('gaps') are a range of sizes and may persist for different lengths of time (Brokaw, 1985). The creation of a gap may directly impact on the microclimate in the immediate vicinity in a number of ways: light penetration to the ground surface may increase; temperature may rise; and humidity may fall (Lee *et al.,* 1990; Lee *et al.,* 1996; Mahid and Jusof, 1987; Kennedy, 1991; Ashton, 1992; Whitmore, 1996). The dynamics of soil processes may be affected by the change in microclimate. Decomposition of organic matter may be affected by temperature and soil moisture (Jordan, 1993). Disturbance created by tree and branch fall is also manifest in damage sustained by seedlings and saplings within the stand (*e.g.* Clark and Clark, 1991).

One of the most important causes of catastrophic disturbance is mechanised timber extraction. This creates disturbance to the canopy in the same way as background disturbance, but, importantly, also directly affects conditions at the ground surface. Harvested trees are extracted by attaching cables to the base of each tree and dragging it out of the stand using heavy machinery such as bulldozers. This can result in the removal of organic matter and in soil compaction in areas over which the bulldozer passes. This has a number of associated problems such as loss of topsoil, soil compaction and lowering of water storage capacity (Basnet, 1992; Malmer and Grip, 1990; Greacen and Sands 1980). The amount of disturbance created depends upon the way in which extraction is organised. Measures such as better planning of skidtrails and directional felling may substantially reduce disturbance (Pinard and Putz, 1996).

Disturbance can result in a complex mosaic of environmental conditions within a forest stand (*e.g.* Denslow, 1987). However within this mosaic it may be possible to distinguish patterns associated with individual disturbance events. For example, it has long been recognised that treefalls create characteristically shaped zones of disturbance. The French word 'chablis' has been used to refer to these shapes (Brokaw, 1985).

2.1.2 Recovery processes

Recovery after disturbance can be split into a number of different stages: seed production and dispersal; seedling establishment; tree growth and mortality. Any of these stages may act as a 'bottleneck' *i.e.* block or slow down the re-establishment of the primary (*i.e.* 'undisturbed') state.

Seed production and dispersal are often intermittent in tropical forests. Years of high production ('mast' years) may be interspersed with years of low production (Ashton, 1989). Once seed has been dispersed predation by forest animals may significantly reduce the number of viable seeds or seedlings in the stand (*e.g.* Itoh *et al.*, 1995). Intermittent seed production means that the timing of disturbance in relation to seed production is important in determining the speed of re-establishment. If disturbance occurs at a time when there are few viable seeds or seedlings then re-establishment is delayed. In contrast, if disturbance occurs at a time when seeds and seedlings are plentiful then re-establishment may progress rapidly. Timing of seed production is particularly important in dipterocarp forests as the seeds of dipterocarps do not show dormancy but start to germinate and grow immediately after dispersal. They may thus be more vulnerable to large scale disturbance such as occurs in logging (Clark and Clark, 1991). However, Pinard and Howlett (1996) found that in a forest in Sabah site conditions (*i.e.* soil moisture, light environment *etc*) were more important bottlenecks for establishment than seed supply.

Intermittent seed production combined with environmental heterogeneity caused by disturbance often creates localised populations of seedlings of the same age within a stand (Ox, 1973; Liew and Wong, 1973; Fox, 1972; Still, 1996; Appanah and Rasol, 1995; Clearwater, 1997). These patches may persist for a few months or years before disappearing due to mortality or (in some but not all cases) growth of the seedlings and saplings into trees.

Initially, the individuals in a seedling/sapling cohort may all be of similar size. This state may persist because the seedlings may be widely spaced so that they cannot deplete resources available to other seedlings. Also, even if seedlings are clustered at points the high mortality rates associated with the seedling phase of development for many species means that inferior competitors are quickly eliminated. This will preserve the size-symmetry in the population (Weiner and Thomas, 1986).

Nutrient availability is an important determinant of early cohort growth. Turnover of organic matter by decomposers in the soil releases nitrogen and phosphorous. The rate of turnover may be determined by temperature and moisture of soils (Majid and Jusof, 1987; Nussbaum *et al.,* 1995). Mycorrhiza may play a critical role in uptake of the nutrients by individuals, particularly for phosphorous. If mycorrhizal inoculum is absent from a site then establishment of seedlings may be delayed or blocked altogether (Alexander *et al.,* 1992; Ahmad, 1996).

As a cohort develops competition between individuals typically becomes more important and the symmetry and similarity in size may break down. In particular, increased competition for light may result in differential rates of growth and mortality (Zipperlen and Press, 1996; Itoh *et al.*, 1995). Competition for this resource tends to be one-sided compared with *e.g.* water or nutrients in that a slightly superior individual can claim a relatively high proportion of the resource.

Individuals in a cohort may assume different forms as they grow. The form of the trees affects ultimate stand physionomy and may also affect the quality of timber produced by the stand. Hallé, Oldeman and Tomlinson (1978) have developed *architectural models* to describe and explain different patterns of tree growth. An architectural model can be conceptualised as a 'growth program which determines the successive architectural phases [that the tree undergoes in its life cycle]' (Hallé *et al.,* 1978). The model captures such details as the predisposition of a species to sympodial or monopodial growth. It does not however determine the precise form that a tree will assume at a point in its development, but rather interacts with the environment of the tree to shape its development. For example, open-grown trees can differ substantially in form from those that have grown within the canopy (*e.g.* King, 1995). Exposure to damage *e.g.* the snapping of a branch may also cause modification to the growth plan.

Many studies have demonstrated differences in the behaviour of species to environmental factors. For example, Fox (1973), Itoh *et al.* (1995), Still (1996), Turner (1990) and Whitmore and Brown (1996) have all demonstrated species differences in the responses of seedlings to factors that are important in disturbance and recovery, *e.g.* response to light or to temperature.

This species-specific behaviour is especially problematical with respect to modelling because of the high species richness of many tropical forest stands. The number of tree species greater than 10 cm dbh found in 0.1 ha can be over 200, compared with 15-20 for the most species-rich temperate forests (Huston, 1994; Argent *et al.*, 1993). Moreover, individuals of the same species can be at low density and highly dispersed throughout the forest. This means that it is very difficult to get a large enough sample size to meaningfully assess the characteristics of individual species.

A common strategy for coping with high species richness is to place individual species into groups on the basis of their behaviour. This is possible because species characters covary *i.e.* certain species characters tend to be regularly associated with certain other characters. For example 'pioneer' species typically exhibit the following characters: absence of seed dormancy, aggressive seedling response to increased light availability, development of large leaves (Whitmore, 1992). In S.E. Asia species of genera *Macaranga* and *Mallotus* often exhibit pioneer tendencies (Primack and Lee, 1991). Different methods have been developed for classifying species on the basis of their behaviour (*i.e.* undertaking *functional* classifications) (Westoby and Leishman, 1997). These are required because it is now recognised that a single 'general purpose' classification is usually inappropriate, and that the way the classification is developed (for example, the characters used) must be related to the purpose of the classification (Gitay and Noble, 1997; Mooney 1997).

2.1.3 Important features of forest dynamics with respect to modelling

The previous two sections have outlined many factors that may, under some circumstances, be of relevance for modelling forest dynamics. However at a single site all factors are unlikely to be important. The precise nature of a model will therefore depend upon site-specific factors. Nevertheless it is possible to abstract a number of criteria from the review material against which models can be evaluated:

- Stand structure should be captured. Stand structure is critical in determining production and dynamics of tropical forests.

- Processes of disturbance to forest stands should be captured. These processes may be of great use in explaining and predicting forest dynamics.

- Processes involved in recovery from disturbance should be captured. The rate at which recovery takes place, and the presence of any bottlenecks can substantially affect dynamics.

- Local variation within the forest should be captured. Phenomena such as competition for resources and disturbance are extremely important in determining forest dynamics. These are local rather than stand-level phenomena in that they operate unevenly across the stand and in that local departures from larger-scale mean values may be especially significant.

- Differences between the behaviour of different species should be captured. Dynamics of the stand is affected by the ability of some species (such as those of genera *Mallotus* and *Macaranga* in some S.E. Asian forests) to respond rapidly and aggressively to disturbance compared to other species (such as those of genus *Shorea*). There is therefore a need to capture these differences in behaviour.

2.2 MODELS OF FOREST DYNAMICS

This section draws on existing reviews of tropical forest models (*e.g.* Vanclay, 1994,1995; Alder 1995; Davis and Johnson, 1987) and agroforestry models (*e.g.* Muetzelfeldt and Sinclair, 1993). The main reason that a new review of the material is required is that this work has several purposes, some of which were not shared by the original reviews.

The first objective of this section is to identify and review models that are useful in the context of sustainable forestry. This is done by using some of the requirements identified in Section 2.1.3 as a set of criteria for evaluating modelling approaches that feature in the literature. Some of these requirements have direct consequences for models. One requirement was that models should be able to capture local variation within forest stands. This is achieved by horizontal disaggregation in a model. Another requirement was that the models should be able to capture differences between species. This is handled with species disaggregation in the models. A third requirement was that stand structure should be captured. This is achieved by using some form of vertical disaggregation.

The second objective is to draw attention to features of models that affect the ease with which models can be implemented. One major factor is the nature of tree representation used in the model. At least four different representation units are possible: all the trees in the stand can be treated together; trees can be split into size-classes; trees can be split into cohorts; trees can be individually represented (Alder, 1995; Vanclay, 1995).

The representation unit is important because different representation units must be managed in different ways in model implementations. For example, some of these units may be created and destroyed in the course of a simulation, while others may remain constant in number. Interactions between units may be fixed and symmetric (in that the same units always interact with one another) or fluid and asymmetric (in that patterns of interaction change throughout the course of a simulation).

2.2.1 Simple empirical models

Simple empirical models are the most widely used class of forest models today, and have a long history of use in temperate countries (Davis and Johnson, 1987; Assmann, 1970). They have also been applied in tropical countries however, to both plantation and natural forests (Mendoza and Gumpal, 1987). The approach averages aggregate statistics of growth and yield produced in experimental plots to provide estimates for different combinations of treatment and forest type. The approach demands that the experimental design have a number of replicate plots for each of these combinations. Another requirement is that the experiments must run for a period of time similar to that of interest for prediction. Philip (1994) describes methods that can be used in more detail.

Most often this modelling approach is used for plantation forests. This is appropriate for reasons related to the way in which plantations are established and managed. Evans (1992) describes practice for plantations in the tropics. In these forests trees within a plot are all of the same species. Often the site is treated before planting *e.g.* by ploughing, so making the soil more homogeneous vertically and horizontally. Fertiliser may be applied, and this may mask variation in intrinsic soil fertility across the site. Trees tend to grow at the same rate, so that size asymmetry may be slow to develop. Slightly inferior competitors are often removed in thinning operations, and this tends to further suppress the development of asymmetry.

For all of these reasons it may be appropriate to lump trees together in a model of a plantation forest. Horizontal variation within the stand is much reduced so that horizontal disaggregation may not be required. Vertical disaggregation is not required because at any point in time trees are similar in size. Species disaggregation is not required because all trees in the stand are of one species.

There are a number of disadvantages to simple empirical approaches. First, the time required setting up and running the kind of growth and yield experiments can be high. For example to make estimates of third cycle growth and yield for TPTI using this approach would require at least 70 years of data. Second, if variability within a forest type is high then a larger number of replicates will be required to obtain a precise estimate of average growth and yield of the forest type. Third, if there are a large number of different forest types or a large number of treatments then the number of plots required can become excessive.

One advantage of simple empirical approaches is that there is less scope for bad model design. More complex models attempt to capture more details of the structure and functioning of the forest stand. While one advantage of using complex models is that they may be more generally applicable one disadvantage is that there is more scope for omission or misrepresentation of important features. For example, more complex models may have to deal with edge effects, while for the simple empirical model it is not a problem.

2.2.2 Size-class based models

Several size-class models have been developed specifically for use in tropical forests under natural forest management (Bertault and Sist, 1995; Mendoza and Setayarso, 1986; Osho, 1990). Size-class models achieve vertical disaggregation by modelling the number of trees in each of a series of size-classes. For example, the structure of a stand at a point in time could be represented by four values giving the number of trees in stem diameter classes 10 to 30 cm, 30 to 50 cm, 50 to 70 cm and greater than 70 cm.

In each time-step, trees undergo transition from one class to its successor or undergo mortality. In the simplest case the number of trees undergoing transition from one class (the *donor* class) to its successor is directly proportional to the number of trees in the donor class (Osho, 1990; Mendoza and Setayarso, 1986). The main problem with this approach is that it does not capture feedback effects, and these may be an important feature of the behaviour of real forests.

More sophisticated size-class models attempt to capture feedback effects. They do this by making transition from a size-class to its successor a function of a variable that captures the cumulative influence of trees outside the size-class. Stand basal area has been used in this way (*e.g.* Solomon *et al.*, 1986).

Other forms of disaggregation can be achieved by progressive splitting of the size-class structure. Horizontal disaggregation can be achieved by splitting the stand into a series of cells. Species disaggregation can be achieved by replicating the size-class structure for different species-groups. For example, Bossel and Kreiger (1991, 1994) describe a sophisticated size-class-based model called FORMIX that has been used to produce growth and yield estimates for lowland forests in Malaysia. The modelled stand of 1 ha is disaggregated horizontally into 4 x 4 gridsquares each 25 m x 25 m. Each gridsquare has associated with it several different series of size-classes, one series for each grouping of tree species used in the model. The species-groupings used reflect the maximum height which individuals of the different species can grow to in the canopy. This means that there are a different number of size-classes in a series depending upon the species-grouping, emergent species having the greatest number (5) and understory species having the least (1).

Size-class based approaches are not always good at capturing dynamics of stands which are not at equilibrium. In the most extreme case of non-equilibrium most of the size classes may be empty and there may be 'pulses' of trees passing through. If there are **n** size-classes then the minimum time for the pulse to be transmitted from the first to last size-class is **n** time steps. When size-classes are empty (as may happen after a logging, for example) some trees may undergo transition from first to last size class in this minimum time. Even if an exceedingly small number of trees undergo fast transition of this kind the consequences are important because of the disproportionately large role of large trees in determining dynamics of the forest stand (Clark and Clark, 1991).

Another problem associated with size-classes has been termed 'aggregation error' (Huston, 1994; O'Neill and Rust, 1979). This occurs when individuals within a size-class differ in some respect to such an extent that it is inappropriate to lump them together. In size-class models the largest size-class usually contains all individuals above a certain threshhold *e.g.* all individuals above 70 cm stem diameter. This means that they can have individuals of a large range of sizes within them. For example, a tree of 71 cm and a tree of 120 cm might occur in the same size-class. The reason that this is done is that large individuals are comparatively rare and using high numbers of size-classes may make models run considerably more slowly. However, as mentioned above, the largest of the large trees can exert a disproportionately big affect on the stand, and any diminution of this may bias model results.

Another problem results from the way the horizontal disaggregation is handled *i.e.* by dividing the plot into a series of square cells. It is difficult to ensure that inter-cellular and intra-cellular processes are handled consistently and realistically. It is intuitive that trees the same distance apart should exert similar influence on each other, regardless of whether they are separated by a cell boundary. However, in models such as FORMIX the two kinds of interaction are handled quite differently, leading to an increased probability that inconsistencies will occur.

Horizontal interaction of representation units can also lead to problems with edge effect. This occurs because representation units at edges will interact with fewer other units. For example, a square cell in a corner may interact with three neighbouring cells, while one in the centre will interact with eight. This problem has been addressed in a variety of ways, none of which is completely satisfactory (Alder, 1995). One common approach is to use 'wrap-around' techniques so that those units on the edge of a plot interact with units on another edge of the plot.

Several different modelling formalisms can be used to design size class models. These include the matrix modelling formalism (Caswell, 1989) and the compartment-flow modelling formalism (Haefner, 1996). Matrix models are relatively inflexible in that it is difficult to incorporate intermediate variables *i.e.* variables other than those used to store the number of individuals in the size class. For example, a variable reflecting light levels experienced by a size-class of trees may be a useful intermediate variable. Compartment-flow models are superior in that they give much better support for intermediate variables.

The implementation of size-class based models does not require a system that can handle creation and destruction of representation units. Interaction between size-classes is also relatively easy to implement because interaction is fixed: a size-class interacts with the size-class below (*i.e.* it receives new individuals from it) and the size-class above (*i.e.* it sends new individuals). The size-classes above are also important in that they may determine the rate at which new individuals are sent to the next size-class.

2.2.3 Cohort-based models

Several workers (*e.g.* Vanclay, 1994; Alder, 1995) have advocated cohort models for use in tropical forests. In a cohort-based model populations of trees that established at the same point in time in a forest stand (*i.e.* cohorts) are modelled. Each cohort is represented by at least two variables, one giving the average size of members of the cohort and one indicating the number of trees that the cohort represents.

Individual cohorts are created and destroyed in the course of a simulation run. Unlike a size-class model in which the number of representation units is fixed, the number of representation units in a cohort model will usually change throughout the course of a simulation. Two refinements of cohort models that are often used are to allow cohorts to split up and to allow cohorts to merge under certain conditions. Splitting of cohorts can capture the development of size-asymmetry (*i.e.* the tendency of members in the cohort to assume different sizes). This behaviour is both important with respect to dynamics and common in cohorts of seedlings or saplings (See Section 2.1.2).

Two cohorts can be merged when their average sizes converge. This prevents a profligacy of cohorts developing through continual cohort splitting. Scheffer *et al.* (1995) have pointed out that technically the name 'cohort' is not appropriate to describe these models, as the 'cohorts' as defined in this approach may notionally contain individuals that established at different points in time. They prefer the name 'super-individual' to 'cohort' in this context. However, this name is more likely to be confused with 'individual-based' and the term cohort model is quite well established, even if slightly misleading.

The amount that the average size of individuals in a cohort changes is determined by the growth rate of members of the cohort. Various functions can be used to determine these growth rates (Alder, 1995). Some of these relate growth to size, while some use size and a competition index of some sort. The number of trees that the cohort represents changes through time due to the mortality of members of the cohort. The mortality rates can be species or size dependent.

Cohort models are similar to size-class models in several ways. They can achieve horizontal and species disaggregation in a similar way to size-class models *i.e.* by using separate representation units for different cells and different species (*e.g.* Vanclay, 1989). This leads to some of the same problems, *i.e.* it may be difficult to model horizontal interaction consistently. They also may produce aggregation error in the same way as size-class models.

The implementation of cohort models is complicated by the fact that the number of representation units in existence changes through time. This means that cohort interaction is harder to manage. When other forms of stand disaggregation are used, *e.g.* splitting the stand into square cells, representation units (cells) may only interact with their neighbours. These are fixed in number and stay the same throughout the simulation. In a cohort model it might be expected that a cohort should interact with all cohorts that are larger than itself. In this case both the cohort identities and the number of cohorts may change in the course of the simulation. However, most cohort models do not model inter-cohort interaction directly.

2.2.4 Individual-based models

Individual-based models (IBMs) are assuming an increasingly important role within ecological modelling in general (*e.g.* Huston, 1994; Judson, 1994; DeAngelis and Gross, 1992; Köhler and Huth, 1998). Moreover, they have been applied to the prediction of long term dynamics in forest ecosystems for over 25 years (Shugart, 1984). Individual-based models (IBMs) capture details of each tree in a modelled stand separately. Typically each individual has one or more variables to capture the size and other characteristics of the individual. The growth and development of individual trees above a certain size is captured.

Inter-tree competition is handled in a variety of different ways in IBMs. Several reviews have been conducted of the various competition indices used in individual-based models (*e.g.* Biging and Dobbertin, 1992, 1995; Dale, Doyle and Shugart, 1985). In general indices are divided into 'distance dependent' and 'distance independent' kinds. In the former kind the precise position of trees is used in calculations while in the latter position information is not used. An example of the former kind is Hegyi's index. The calculation of this value for a 'subject' tree involves summing the value of the distance weighted size-ration (dwsr) calculated for all 'competitors' (*i.e.* all trees within a certain distance of the subject tree). The dwsr is found for a competitor by dividing the subject tree stem diameter by the competitor diameter then multiplying by the inverse of the distance separating the two. The 'gap' models described below provide examples of distance-independent competition measures.

Many models explicitly model the transmission of light through the canopy. Gap models typically employ horizontal disaggregation based on cells to do this (Botkin *et al.*, 1972; Shugart, 1984; Friend and Shugart, 1993). In this method the stand is split into a series of square cells. Trees in the same cell are sorted into vertical layers on the basis of height. The foliage in each layer is calculated by summing the foliage of all trees assumed to contribute to the layer. Transmission of light through the layer is then modelled and the light available for trees in the different layers is calculated. Individual tree growth is then determined using an equation based on light absorption. These calculations can all be done in the absence of precise information on tree position and for this reason this kind of growth model is sometimes called 'distance independent'.

Other IBMs that explicitly model the transmission of light through the canopy make use of tree position information. This involves three-dimensional disaggregation of the forest canopy (*e.g.* Wang and Jarvis, 1990). These models are able to take account of latitudinal and seasonal variation of solar inputs to the forest. One problem with the latter approach when applied to natural forests is the irregularity of the forest. Often trees grow opportunistically. For example, the stem of a tree may grow so as the crown can exploit gaps in the canopy. This can lead to a bent shape such that the position of the crown is difficult to predict from the location of the stem. Given this irregularity, it is very difficult to model the location of foliage in 3-dimensions. The extra detail may increase the time it takes for the model to run and the number of parameters required but may have little affect on the accuracy or precision of estimates produced by the model (Sianturi, 1997).

Most IBMs use stochastic functions to handle mortality. There are two basic methods. In the first, background mortality events such as treefalls are not explicitly modelled. Instead each tree in a modelled stand is assumed to have a specific probability of undergoing mortality in a time-step. Monte Carlo methods are used to determine the trees that actually undergo mortality in a particular year. (Botkin *et al.*, 1972; Shugart, 1984).

Recently a second approach has been used in which background mortality events such as treefalls are explicitly modelled. In this approach each tree in the stand has a probability of initiating a tree fall event. Each treefall is captured by projecting a shape onto the forest floor. The shape is related to the dimensions of the tree that falls. A mortality probability for other trees in the stand which are inside this shape then is then determined, and trees which actually undergo mortality are determine using Monte Carlo methods as before. (Köhler and Huth, 1998; Kurpick *et al.*, 1997). As discussed in Section 2.1.2, this is more appropriate as up to 75 percent of mortality can be secondary mortality.

One of the main advantages of using IBMs is that they avoid aggregation error (Huston, 1994; O'Neill and Rust, 1979). They can also avoid problems associated with cellular horizontal disaggregation (though of course, 'gap' models may still suffer from some of these problems). In particular they are good at handling the interaction of spatially defined phenomena such as skidtrails and standing trees, or tree disturbance zones and standing trees. Given that interest is currently focusing on areas such as reduced impact logging, this is an extremely important advantage.

IBMs can suffer from edge effects in the same way as spatially-explicit size-class models and cohort models. The situation is even more difficult to handle because of the way in which spatially defined interaction takes place. In models that use a cellular grid, edge effects can be compensated for by mapping edges of cells on the periphery of the plot to the opposite edges of the plot. This is comparatively simple and need only be done once. In contrast, in an individual based model interaction across edges is more difficult to capture and involves a series of calculations every time step (Alder, 1995).

The main disadvantage with using IBMs is that data on tree positions must be recorded in forest mensurations. However, the main resource involved in collecting this data is labour - no expensive equipment is necessarily required. This means that the disadvantage is not so severe. There may also be scope in future for using techniques to simulate tree locations (*e.g.* Diggle, 1993)

Implementation is complicated by the fact that representation units sometimes need to be *sorted* before calculations can take place. For example, in IBMs in which transmission of light is modelled units must be processed in a specific order such that a unit receiving light is processed before the unit that it transmits the light to. In other model classes sorting of representation units is not required.

2.2.5 Use of models in the context of sustainable forest management

Most of the models described in Section 2.2 are designed to capture feedback effects such as reduced competition created by removal of the large trees. However, most of them will struggle to cope with investigations in which subtle horizontal interactions are of importance. This is the case in investigations of Reduced Impact Logging, where the option of planning skidtrails must be compared with the option of not planning skidtrails. Capturing this involves being able to calculate how many trees will be inside skidtrails under each option. Even for those models in which horizontal disaggregation is achieved by associating units of tree representation such as size-classes or cohorts with gridsquares in a grid, this will be problematical.

When using individual-based models in which tree-co-ordinates are explicitly represented these issues may be more easily resolved. The disturbance created by each tree can be calculated and overlap in disturbance determined. Individual-models are also not prone to problems from aggregation error.

Individual-based representations may not be good at capturing the early stages of forest recovery, so that other representations must be considered. This is because they are not efficient when large numbers of individuals share the same properties. This is likely to be the case at seedling establishment (Section 2.1.2). More aggregated units of tree representation are appropriate in such circumstances. These can be associated with gridsquares in a grid that covers the stand to achieve horizontal disaggregation. Of size-classes or cohorts, the latter are likely to be the superior units of representation. This is because certain size-classes of seedling are frequently not represented in forest stands *e.g.* because of intermittent seed production in certain forests (Section 2.1.2). As stated in Section 2.2.2 empty size-classes can cause size-class representations to exhibit pathological behaviour.

2.2.6 Common features of forest simulation models

The framework that is envisioned in Section 1.5 supports simulation models for predicting long term dynamics of tropical forest. To determine the nature of the support provided by the framework, it is necessary to consider potentially exploitable characteristics of the simulation models likely to be used in the framework. Several features of the simulation models discussed in this section are relevant:

1 Use of Systems Dynamics concepts.

All of the models described are simulation models so that all use concepts such as model state, state variables and intermediate variables.

2 Logical grouping of variables *i.e.* use of 'representation unit' abstraction.

In all of the models described (aside from the simple empirical models) an aggregate representation is used in which the stand is represented by one or more units. These units are individual trees, cohorts or size-classes. In each case the unit itself possesses Systems Dynamics attributes such as state variables or intermediate variables. For example, individual trees possess attributes of stem diameter, height crownpoint *etc*. Cohorts may possess attributes of mortality, ntrees (a count of the number of trees represented by the cohort). Free-standing variables are much rarer. It is useful for purposes of displaying and manipulating data to group this data.

3 Sharing of attributes by representation units.

Representation units often share attributes but not attribute values. For example, all trees may be represented by the same set of attributes, and all cohorts may be represented by the same set of attributes.

4 Destruction and creation of representation units.

Some kinds of representation unit (individual trees and cohorts) are regularly created and destroyed in the course of simulations.

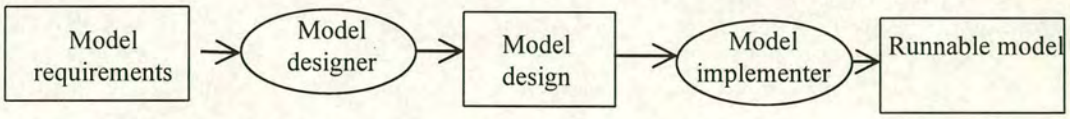
2.3 STRATEGIES AND SYSTEMS FOR DEVELOPMENT OF ECOLOGICAL MODELS

Many different strategies for model development are currently in use. In common with other design processes, modelling is usually iterative in nature in that it is not linear but involves revisiting areas of design that ostensibly precede the current design phase. Further there is now a growing realisation that a single design will not be optimal in all circumstances. There is therefore a need for systems to be able to handle change in models to meet particular circumstances. This will usually involve facilitating forms of sharing and reuse of functionality between models, as it will be inefficient to create new models from scratch. This section discusses stages in the model design process, the need for flexibility in modelling systems and how this flexibility is catered for in some systems.

2.3.1 Stages in the model design process

Many workers have attempted to produce schemes outlining various logical stages in the modelling process (*e.g.* Bossel, 1994; Vanclay, 1994; Innis and O'Neill, 1979). The approaches are reviewed in Haefner (1996). The part of the process that is here referred to as *model realisation* begins with series of articulated requirements and ends with a runnable model. It is possible to represent these stages in a Data-Flow diagram (Figure 2.1).

Figure 2.1: Data Flow Diagram showing the various stages involved in the realisation of a model



Model requirements will typically specify the generality, accuracy and precision required from the model. For example, the model requirements may specify that the model must give predictions of growth and yield of natural forest for a single silvicultural system. Alternatively the requirements may specify that the model gives predictions of growth and yield of natural forest under different logging intensities, or growth and yield of forest plantation under a single management system. The model designer will typically develop a design using *conceptualisations* from a particular modelling paradigm. For example, 'state variable' is a conceptualisation from the Systems Dynamics modelling paradigm (Caswell *et al.*, 1972; Bossel, 1994). State variables are variables in a model that have a number of special properties. They:

- describe the state of a modelled system at a point in time;
- must be initialised at the start of a simulation;
- change through increment or decrement in the course of a simulation;

A single model may employ several state variables, each of which captures a particular detail of the modelled system. More formally the conceptualisation of 'state variable' may be thought of as a *class* (*i.e.* a template used for creating individual model state variables) which can be *instantiated* in one or more model designs by specifying particular relationships and properties. For example, a state variable could be assigned the name 'stem-diameter' and the units of 'cm' in a model design.

The end product created by the model designer is a model design *i.e.* a specification of all the decisions taken in the design process. Model designs may be formal or informal, diagram-based or text-based. Formal designs are *complete* in that enough information is present so that arbitrary decisions taken by the implementer will not affect results produced by the runnable model. Five different classes of design can be recognised:

- Informal, text-based designs. These designs are comprehensible but may be unambiguous or incomplete. Accounts of models presented as papers in scientific journals are examples of this kind of design (although these accounts may draw on diagrams or equations they tend to be primarily textual). The main disadvantage of this kind of design is that often the vital details are omitted so that it is impossible to recreate the model on this basis alone. For example, parameter values may be omitted or detail on how a particular part of the model functions.

- Informal diagram-based designs. These designs are comprehensible but may be ambiguous or incomplete. For example, state-transition diagrams, entity-relationship diagrams, data-flow diagrams and inheritance diagrams can all be used to express some details of models based on the object-oriented paradigm (Rumbaugh *et al.*, 1991). These designs are similar to informal text-based designs in that they must be implemented by human computer programmers as decisions will often need to be taken on details not specified in the design. They may for example, indicate that 'tree' is a class of 'plant', and possesses the method 'grow' but give no information on how this should be implemented.

- Formal, text-based designs. These designs differ from the previous two kinds in that they are typically complete and unambiguous. These often employ a language with special vocabulary, syntax and semantics for expressing design details. For example, Muetzelfeldt *et al.*, (1989) describe how such languages can be based on the syntax of the logic programming language Prolog, while Maxwell and Costanza (1997) describe a language used to create a set of objects used in model simulations. Abel and Niven (1990) describe how a language called 'z', which is based on set theory, can be used to develop a model of the ecology of a species of octopus. This type of design is usually complete and so unambiguous. Moreover, in the case of Prolog representations, a lot of useful functionality can readily be incorporated into tools for model design. Such tools may be used to check the structure of a model for correctness, or to allow users to interrogate the model to determine its structure (Muetzelfeldt *et al.*, 1989).

- Formal, equation-based. In some cases it may be possible to completely describe a model using one or more equations. For example, the famous Lotka-Volterra predator prey model can be described using two differential equations (*e.g.* Renshaw, 1991). This formalism, while simple, is not good for expressing the structure of many modern models. For example, it cannot be readily adapted to capture details of the disaggregation patterns used in a model.

- Formal, diagram-based. The designs created in a modelling environment such as Stella (High Performance Systems, Inc., 1992), ModelMaker, (Cherwell Scientific, 1998), Powersim (ModellData, 1995) and AME (R. Muetzelfeldt, *pers. comm.*) are examples of this kind of design. Diagrams in these systems are used to specify compartment-flow models. Different shapes are used to represent different kinds of concept. For example, boxes may correspond to state variables, while circles may correspond to intermediate variables. One disadvantage of using this approach is that it can be difficult to express some features that can readily be expressed as algorithms in a text-based design. For example, it is quite natural to express the nature of a logging operation as an algorithm *e.g.* 'remove all trees on slopes that are less than 15 % and that are over 50 cm in stem diameter'. Expressing this in a diagram-based formalism is difficult if not impossible.

Model realisation (the process of expressing the model in a form that enables inferences to be drawn about its behaviour) involves the implementation of the model in one of several ways:

1. Manual coding of model design. In this approach a human computer programmer is responsible for taking the design and producing an implementation. This involves production of source-code in a computer language such as BASIC, Pascal, C, C++ or FORTRAN, compilation of the program, testing and debugging. The ultimate output of the process is a stand-alone program (*i.e.* an application) that allows users to conduct simulation runs with the model.

2. Automatic source code generation. In this approach the design is passed to a special software program called a *code generator*. This then produces source code that can be compiled to produce a runnable model. In the approach described by Muetzelfeldt *et al.* (1989), a concise specification is passed to a code generator (written in Prolog). The code-generator then produces the high level source-code (for example BASIC code) necessary to implement the model. The code can then be linked and compiled in the normal way to produce a runnable model. The advantages of this approach are that the production of the first runnable model is much faster and also that as the process is automatic errors are less likely to occur so that less time testing and debugging is required.

3. Automatic interpretation of model design. In this approach the model design is passed to a special piece of software called a *model interpreter*. The interpreter then allows users to run the model. In this case the bringing together of the design and the interpreter creates the 'runnable model'. It is not a stand-alone program in the way that runnable models produced by the previous two methods are. The approach also differs from the previous two in that the production and compilation of source-code does not take place. This approach is used in the formal diagram-based systems discussed in the last section *i.e.* Stella, AME and Powersim. The advantage of this approach is that a single piece of software must be used (in the previous approach at least three must be used - code generator, compiler and linker). While in theory this should not make a difference when other pieces of software are automatically invoked, in practice it does, as the process is more robust. This enables stages of the model realisation process to be more tightly integrated.

2.3.2 Handling changes to model design through time

In the course of the modelling activity, models are inevitably changed to take account of new circumstances. The rationale for making these changes and the way in which the changes are organised vary widely. Such factors can have major implications for the optimal planning of modelling activity, and it is for this reason that they are discussed in this section.

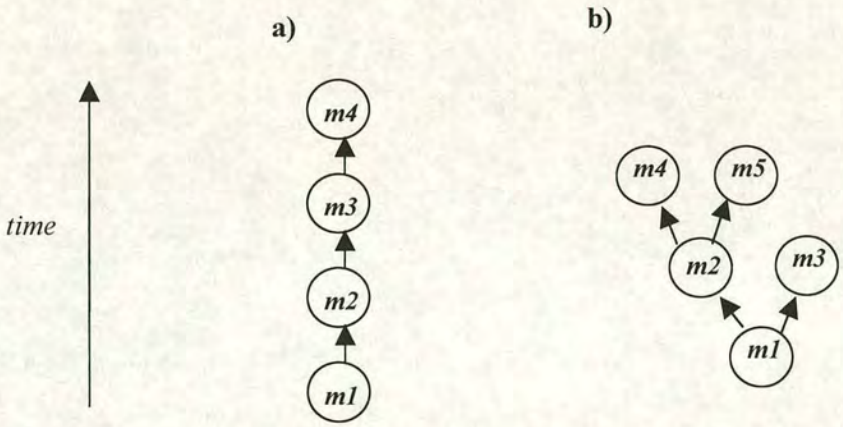
Reasons for changes in model design fall into two categories:

1. 'Refinement' of model design through increase in complexity. Most schemes for model development specify an iterative process of some kind, with model results being compared with real data (validation) and the model being changed to reflect any shortcomings that are found (Haefner, 1996; Bossel, 1995; Kimmins, 1996; Vanclay, 1994). Sometimes these changes may involve parameterisation but often they will involve increasing the complexity of the model design. For example, model designs may be progressively altered to include nutrient dynamics, water relations and response to increased CO_2. It is often implicit in these approaches that the end-goal of refinement is a single representation of the forest that is useful for many purposes. This model development rationale has been strongly criticised on two grounds. First model representations can never be complete (Bossel, 1994; Haefner, 1996; Vanclay, 1995). It follows that the quality of a model can be assessed is in relation to a specific objective. They can therefore never be completely 'general purpose' (Rothenberg, 1993). Second additional complexity often does not yield improvements in model prediction, but actually makes models more difficult to use. The lack of improvement in prediction may arise because in any one situation only a subset of the factors represented in the model may actually influence the observed behaviour of the model. This means that complex models often require parameterisation, which may be expensive or otherwise difficult and which adds nothing to the quality of the model predictions (Passioura, 1993; Kimmins, 1993).

2. New requirements for the model. A single model should be designed for a purpose and will only be valid for a certain range of conditions. This fact is very widely acknowledged in the modelling community (*e.g.* Bossel, 1995; Kimmins, 1996; Vanclay, 1994). In the context of sustainable forest management new requirements may arise for a number reasons: there is a need to adapt the modelling for a different forest type; there is a need to adapt the modelling for new silvicultural systems or to take account of new management goals; there is a need to make use of new understanding of the biology of the system. As discussed in Section 1.2, the first of these two are inevitable when attempting sustainable forest management.

While the various rationales for changing model designs are often discussed, the consequences of adopting a particular development paradigm are not. These are important as they can significantly influence the effectiveness of a modelling system. In general, the way that changes in model design are organised as part of modelling activity is very variable. However two extremes of behaviour that are here called 'centralised' and 'collaborative' modelling can be recognised. A diagrammatic representation of these two kinds is given in Figure 2.2.

Figure 2.2: Different ways in which model development through time can be organised. Circles represent different versions of a model. a) shows 'centralised' modelling in which a single version is in use at any point in time, while b) shows 'collaborative' modelling in which different versions may co-exist at the same point in time.



In centralised modelling, the model design evolves in a linear fashion, and a single version of the model is maintained at a central location. Centralised modelling is a natural way of handling change due to model refinement *e.g.* adaptation of models to include nutrient dynamics. There is no need to retain previous versions of a model, because the successor is regarded to be superior in one or more respects.

The development of the FORMIX series of models provides an example of centralised models. These were all developed by workers at the Environmental Systems Research Group of the University of Kassel. Three versions were produced FORMIX, FORMIX2 and FORMIX3 (Bossel and Kreiger, 1991; 1994; Huth *et al.*, 1997). The main change between versions 1 and 2 was the introduction of species disaggregation, while changes between version 2 and 3 focussed on the way that growth was represented.

In collaborative modelling, modelling activity is not linear. Different modellers may be responsible for adapting a model to a particular set of circumstances. Collaborative modelling is a natural way of handling change in model design that adapts models to local situations, as collection of data and adaptation of models to suit a particular forest is best undertaken close to the forest in question. It is also useful for parallel development of models so that many different problems can be addressed at once.

The modelling activity that resulted in the production of HyPar is a good example of collaborative modelling. Initially a gap model called ZELIG (described in Urban *et al.*, 1991) was joined to an ecosystem process model called FOREST-BGC (described in Running and Coughlan, 1988) to create a model called HYBRID (Friend and Schugart, 1993). This model was then joined to a crop model called PARCH to create a model which became known by the name HyPar (Lawson *et al.*, 1995). In parallel with this PARCH was also joined to the model MAESTRO (Wang and Jarvis, 1990), which was itself a descendant of pre-existing models developed for use in New Zealand and the USA (Lawson *et al.*, 1995)

A general consequence of pursuing collaborative modelling is that the facilitation of sharing and reuse of model functionality becomes much more important. This is because if the system is to work efficiently then models and model fragments will be exchanged and reused by a community of users.

2.3.3 Sharing and reuse of modelling functionality

It is often the case that different models have certain features in common, either by accident or design. Models may be developed to satisfy similar objectives and designed to be used by a similar user-group. This will typically lead to convergence in features, even when models are developed independently.

Two types of functionality are often shared between models that are developed as part of a collaborative effort: *interface* and *content*. Implementation interface is used to structure the way that humans interact with a model. In modern modelling systems users typically interact with a Graphical User Interface mounted on a PC. The interface will usually make use of *metaphors* (conventions that draw on experience users possess from a different context) and *idioms* (other useful conventions). For example, modelling software developed for the MS Windows™ operating system often uses an idiom whereby the main window of the application has a menu bar which has the menu command 'File' under which options for loading files and exiting the software can be found.

The content of a model design details how the generic conceptualisations used to build the model such as 'state variable' are used in the model to capture details of the system being modelled. For example, the information that trees are represented by the state variables 'stem-diameter', 'height' and 'crown-radius' is part of the content of a model. Model designs that are developed in parallel to satisfy similar objectives will often share a large amount of content. For example, models that capture different management options may use the same set of data to represent trees in the stand, and the same equations to specify the calculation of tree attributes such as height and volume.

Collaborative modelling efforts vary in the degree which they facilitate and encourage sharing and reuse of model interfaces and content. In some there may be specific provision to support sharing with minimal effort, while in others it requires fundamental re-implementation of shared features. The various mechanisms by which sharing and reuse are achieved are reviewed in the following sections.

Sharing of interface

The main advantage of collaboration in which all models use the same interface is that the time taken for users to learn how to interact with new models is much reduced. In general the tasks that a model will be used for are fairly standard, even if the details of how the tasks are performed may differ considerably between models. Typically they are used to:

- Initialise models with data concerning the state of the modelled system at the start of a simulation;
- Initialise models with any other data required;

- Start and interrupt model simulations;
- Produce visualisations of the stand in the course of a simulation;
- Graph time-series results;
- Output results to file;
- Compare different experimental treatments;
- Conduct sensitivity analysis;

Sharing and reuse of a model interface is possible when models are standardised in some way. Usually a specification of how standardisation will occur takes place before the models have first been implemented. In this case all models will be constructed to conform to a pre-existing standard. However alteration of pre-existing models to conform to a standard so that they can use a pre-existing interface is also possible (*e.g.* Knox *et al.*, 1994).

Sharing and reuse of model interfaces can take place in three ways:

1. Partial standardisation of design content. If models share at least some of the same content then the construction of a standard model interface becomes straightforward. This is because detail specific to a single model design does not need to be stored and accessed by the interface. For example, in a set of models trees may be represented by the same data items of stem diameter, height and crown-radius. As the model representation of trees is the same across all models, the modelling interface does not need to consult a file where data on the representation of trees used in a particular model is stored. This approach is particularly common when realisation of new models is undertaken by a human computer programmer who creates new source-code. For example, the three versions of FORMIX described in the next section share a very similar interface. This was not created from scratch each time but by joining pre-existing interface code to code for the new model.

2. Standardisation of design conceptualisations. If the conceptualisations used to define model content (*e.g.* state variable) are standardised across a set of models of differing content then it is sometimes possible to construct an interface that can work with all the different models. In this approach the interface consults a model design detailing how different conceptualisations are used in a particular model. This is the approach taken in modelling environments such as Stella and AME.
For example, if all models adhere to the same conceptualisation of 'parameter' then it is easier to construct a generic interface tool that allows users to change parameter values. This can be done by:

   a) defining values (such as *e.g.* growth coefficients or logging characteristics) to be instances of 'parameter' in the model design;

   b) reading values identified as being 'parameters' into the model interface;

   c) manipulating such 'parameters' in the modelling interface in a manner consistent with the conceptualisation. For example this could mean allowing users to change these values at any point in a simulation while preventing the user from changing other values used in the model.

   These steps mean that the modelling interface can provide a high-level of functionality (in terms of support for handling parameters) for models which may vary a lot in content.

3. Translating models so that they can be used with different interfaces. In some cases it may be possible to 'translate' model designs so that they become compatible with an interface despite not sharing the same design conceptualisations. This is useful because:

   - It may be desirable to use specialised interfaces to perform particular tasks. For example, some modelling tools may have a high level of support for visualisation of simulation data (*e.g.* have tools for construction of profile diagrams of forest stands) while others may have a high level of support for model construction *e.g.* Stella.

- It is difficult to ensure interface standardisation with respect to conceptualisations. Interfaces may be constructed by different people for different tasks at different times.

Translation could involve, for example, substituting terms. For example, what is known in one system as an 'object' might be known in others as a 'sub-model'. Although translation of this kind is not currently used in the domain of ecological modelling, a lot of effort is focussed on developing technologies to support this kind of activity (*e.g.* Neches *et al.*, 1991).

Sharing of model content

The main benefits of adopting a modelling system that supports sharing and reuse of model content concern the ease with which new models can be created. Sharing and reuse are also useful for collaborative modelling - different workers can exchange models or parts of models that they have created. Another advantage is the ease with which models with some shared content can be compared with one another: if models are developed separately then there will inevitably be problems with inconsistencies such as different model designers or implementers using different names for the same quantity.

Sharing of model content is a non-trivial exercise as very often parts of a model are inter-dependent. For example, an algorithm for calculating the diameter increment of an individual tree may use inputs of stem diameter and crown-dimensions. This means that if it is to be used in some model then both stem diameter and crown-dimensions must be represented in the model. This may not be the case in all models, so that the mechanism for sharing the algorithms must check that an algorithm is compatible with the rest of a model design.

Modelling systems can be divided into two kinds: those in which model content is packaged specifically to support interchange between models; and those in which model content is not packaged to support interchange of model content between models. In the former packages of design content are often referred to as *modules* so that the systems that use them can be called *modular* systems.

Non-modular systems

Sharing and reuse of model content is difficult in non-modular systems. There are a number of problems: often models from which it is desired to lift and use content are written in different languages or designed to run on different platforms; often the model design is informal and incomplete so that the implementer must work with the model at the level of source code; there are no groupings of design elements to simplify the process of plugging and unplugging fragments of model content.

Some of these problems are illustrated by the experiences of a single three year agroforestry research programme undertaken by a group based in Edinburgh (Lawson *et al.*, 1995). This group undertook no less than three separate model 'linkages'. In each of these linkages two pre-existing models were joined. Mobbs (1995) describes the way this took place for one of these linkages in which a forest model (Hybrid) was linked with a crop model (PARCH). Initially this joining took the form of running the tree model, storing output from the model in a file, then running the crop model using the tree model data as input (though the inputs and outputs of the models had to be altered to make the models compatible). As Mobbs points out, this meant that there was no feedback from the crop to the trees. Subsequent to the first attempt at linking the models an attempt was made to reimplement the models so that they could run concurrently. This was complicated because the models were designed to run on different platforms (UNIX and DOS), were written in different languages (FORTRAN 77 and Visual Basic for DOS) and one (PARCH) had a graphical user interface while the other did not. Initially an attempt was made to keep both models separate and implemented in their native language. This involved changing Hybrid to a PC platform, then arranging so that a tree sub-routine within PARCH called the Hybrid executable. However, this was found to exceed the memory capacity of the PC. Finally, PARCH was translated to FORTRAN, and the two models were combined at the level of source code. One disadvantage of the latter was that the graphical user interface of PARCH was lost.

Sometimes claims concerning modular program structure are made on a basis that is disingenuous. For example, it is sometimes argued that any computer program that uses constructs such as 'functions' or 'procedures' or 'subroutines' is modular. This is because each function is ostensibly a piece of code with a recognisable boundary and a defined set of inputs and outputs (*i.e.* the *arguments* to the function). The argument made is that this structure means that it is easier to manage substitution of one function for another, so that the system satisfies the conditions for modularity. However, it is frequently the case that a function modifies data that is not in its argument list. In the past programs often made use of data with global scope *i.e.* data that can be read or written in any part of a program (*e.g.* common blocks in FORTRAN). This data can be both read and changed within a function without appearing in the argument list. This means that the content of a function must be inspected on a line-by-line basis to check for dependencies, so that functions can only be said to be modules in the weakest sense of the word. More sophisticated programming paradigms may make more use of encapsulation *i.e.* restricting access to data so that any changes take place in a structured fashion.

Modular systems

Modular systems simplify sharing and reuse by placing permanent, well-defined boundaries around pieces of model content. Each of these boundaries encloses one or more model design elements. When modules contain more than one design element then the group of design elements within a single module are logically related in some way. Different kinds of module are used in different systems. In some a module corresponds to an algorithm (*e.g.* an equation for calculating tree height on the basis of diameter); in others it is a definition of an entity that is captured in the model (*e.g.* a tree); in others it may correspond to a 'sub-model' (*e.g.* water relations sub-model of a forest stand model). All of these can be classified according to whether the module specifies representation data used in a model, algorithms used in a model or a set of logically related data and algorithms.

Modules simplify interchange of model content because they define permanent boundaries within models. Every time content is interchanged an effort must be made to deal with dependencies (*e.g.* if an algorithm for calculating tree diameter increment on the basis of competition index is to be incorporated into a model then tree competition must be represented). In a non-modular system this may involve making checks to ensure that the requirements for every design element are met before incorporation of a model fragment. This may sometimes involve a painstaking search of thousands of lines of source code. When fragments of model content are permanently defined two options become possible:

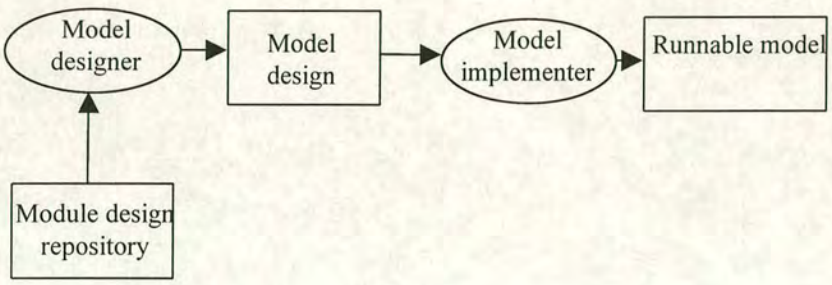
- Structuring fragments so that sets of fragments share the same requirements and hence members of the same set can be interchanged. Often this is done so as to minimise the requirements for modules so that they can be used in a wide variety of circumstances. This often involves 'hiding' (*i.e. encapsulating*) the contents inside the module in such a way that modules of diverse content can substitute for one another. Encapsulation is a well-established and respected technique for increasing sharing and reuse of computer software components (*e.g.* Blair, 1994; Booch, 1994; Goodland, 1995).

- 'Tagging' the fragments with information concerning the requirements that must be met for their use. In this case interchange will involve comparing requirements for the module with the rest of the design to determine compatibility. Muetzelfeldt (1995) describes this approach. He shows how information on the requirements for a module can be represented and manipulated using the Prolog computer programming language. This can aid the process of model design in three ways: the designer can select a module, then be advised if it is compatible with the rest of the current model design; the designer can ask the computer to list all modules that are compatible with the current design to ease the process of decision making; and the designer can be prompted to supply the additional information required by a specific module.

Both of these approaches mean that time-consuming analysis of the fragments need only be done once when the module is defined instead of every time an interchange is required.

Topologies of modularity can be classified according to the way that module *repositories* are used. A repository is simply an entity, either conceptual or physical, that stores modular material. Figure 2.3 shows one of the most important schemes of modularity in use in ecological modelling systems. In this scheme there is only one kind of module repository, the module design repository. The designer selects module designs from the repository and these are directly incorporated into the model design. The full design is then passed to the implementer that creates a runnable model. There is no use of other module material within the system. For this to work the model designs must be complete and comprehensible by the Model implementer (whether this is a human being or a piece of software).

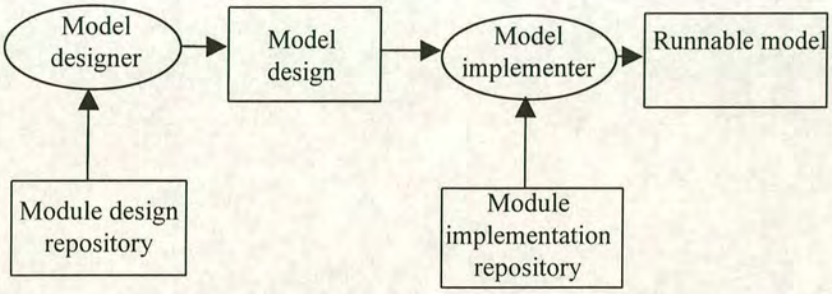
Figure 2.3: Modularity using a single module repository



This topology is often used in systems that utilise formal, complete model designs with automated implementation. For example, the ModelMaker, Powersim and AME modelling environments use this form of modularity. In all of these systems modules are contained in files that can be 'loaded' into a model. The loaded modules can be inspected and edited in the same way as other model content. Edited modules can then be saved. New modules can also be created as part of the normal design process. This form of modularity differs from the others described in this section in that modules are completely transparent in that their content is apparent to the model designer without any need for elaborate schemes of module documentation or inspection of source-code.

The second common topology for incorporation of modules is shown in Figure 2.4. In this scheme there are two kinds of module repository: the module design repository and the module implementation repository. The designer chooses which modules to incorporate based on information in the module design repository. These choices form part of the design transmitted to the model implementer. The implementer then takes the appropriate module implementations from their repository and incorporates them in the runnable model.

Figure 2.4: Modularity using two module repositories



The module design repository can be formal or informal in the same way as the model design can be. Often informal, text-based descriptions are used. For example, when using a software library such as that provided by an operating system (*e.g.* the MS Windows Application Programmers Interface) there would usually be a text description of the different functions, their arguments and the way that each is designed to be used. Often this documentation will be on-line. At the very least the module design must indicate the name of the module and what it does.

The module implementation repository stores modules in a form that is closer to the form in which they will eventually be used. The main advantage of using this approach is that the production of a runnable model from a model design can proceed more quickly. The module implementation repository contains modules in either source-code form or in binary form (but the same repository can only store one of these forms). It can be a Dynamic Link Library (a DLL), a source-code library or an import library. One disadvantage of this is approach is that effort must be expended to ensure module design and module implementation repositories remain consistent at all times. This is the familiar, onerous problem of documentation of computer software.

In any scheme where an implementation repository is used the implementer has the task of joining the thing that invokes and manages the modules to the module implementations. This can be done in two ways: static linking (when implementations are stored as source-code) and dynamic linking (when the module implementations are in binary form). Static linking is usually undertaken as part of the compilation of source-code to produce executable code. In contrast dynamic-linking takes place when a piece of executable code is running (*i.e.* without any extra compilation of source-code).

Both static and dynamic linking define *module interfaces* for the purposes of compilation. These consist of information on the module that must be present when the code unit that invokes the module is compiled to allow it to use the module. For example, for a code unit to be able to link to a function it needs to know the name of the function and some information on its inputs and outputs. The input and output information required comprises both the total number of arguments and the type of each argument. (Collectively this set of information constitutes a *function prototype.*) If one of the arguments is an abstract data type (such as 'tree' or 'gridsquare') then this type must be defined when modules and the code unit that invokes them are compiled. The representation data used for modelled entities may also form part of a module interface when a module contains a class definition for the modelled entity. In order for a code unit to access member functions associated with the class information on the class member data and member function prototypes must be present when the code unit that uses the module is compiled.

Dynamic linking contrasts with static linking in that it is possible to alter module selections without recompilation of either the modules or the code unit that invokes them. This means that the process of selecting modules can be more tightly integrated with the process of running models. For example, it may be possible for a user to change the modules used in a model between runs of the model.

A special form of dynamic-linking was used as the basis of modularity in the LandLord system (Duncan Heathfield, *pers. comm.*). In this system a series of cells is used to model a surface of some kind -in the past the system has been used to represent Mediterranean landscapes. A separate model is associated with each cell. These models are created by repeatedly producing instances based on the class definition stored in binary form in a module implementation repository (a DLL).

There are two main problems with dynamic linking. First, model design is effectively split into two stages. In the first, which is usually carried out only once, a 'framework' consisting of all the module interfaces is specified. Typically interfaces will be designed to be as 'generic' as possible. As mentioned previously, often the definition of interfaces involves deciding on the representation of modelled entities. For example, this stage might involve deciding on a generic tree representation using data of location, stem diameter and height. Using this approach it is not possible to change the detail of the representation *e.g.* by adding information on crown dimensions.

The second stage will typically take place when the model is actually running. A different person to the original programmer will often undertake this stage. The second person will have a well-defined problem to solve, while the first person will be trying to specify a generic solution. One problem with this is that the original programmer may not be sufficiently aware of specific issues to produce a useful framework; the second person may be forced to constrain their design to fit the sub-optimal framework produced by the original programmer.

The second problem with dynamic-linking is that it is not good at providing support for choosing different modules on the basis of their content. This may have arisen because it has been primarily designed for applications other than ecological modelling. For example in a computer operating system a module stored in a DLL may be used to handle interaction between a computer and a printer. In this case only one module needs to be used *i.e.* that applicable for the printer that is currently connected to the computer. The module must perform certain standard tasks, but the printer user is usually completely oblivious to how it performs them without any ill effects. The problem of supplying information about how a module works in order that an informed choice of module can be made does not occur. This module may be updated to increase functionality, but only one module will ever be applicable at a time.

In contrast, in ecological modelling, the content of a module is critical for making design decisions. Consider the case of tree growth. In areas that do not experience seasonal water restriction it is unnecessary to model the affect of water limitation on growth (this leads to redundant and expensive parameterisation). Two modules for capturing tree growth may be present, one capturing water limitation, the other not. To make the appropriate choice the designer must know the content of each. As the module implementations are stored in binary form it is not possible for the person undertaking selection to determine what they do by inspection. For this reason documentation of modules must be extremely rigorous, and a rigorous procedure must be adopted to ensure complete correspondence between the module and its documented description.

Modular systems which make use of source-code repositories may be divided into two kinds: those in which source-code is compiled and linked without any alteration; those in which source-code is modified before compilation and linking. McGown *et al.* (1994) describe a system called APSIM for creating models of agricultural crops in which source-code is drawn from a repository, compiled and linked without modification. In this system the modules contain functions and the module interface is the function prototype. However, because source-code is not modified it means that modules that are substituted for each other must share the same standard prototype. For example, this means that routines in alternative 'crop' modules must have the same names and the same inputs and outputs.

In other systems source-code is modified before being statically-linked in the process of model creation. In these systems 'modules' could contain, for example, the definition for a particular class of object. Several of these systems have been specifically developed for ecological modelling (Lorek and Sonnenschein, 1998; Baveco and Smeulders, 1992; Lhotka, 1994; Larkin *et al.*1988; Costanza and Maxwell, 1997). The model implementer is a human computer programmer who customises the generic (*i.e.* foundation) classes to suit their purposes. This customisation may take the form of adding new member data to class, adding new member functions to the class or changing the way in which pre-existing member functions are carried out. For example, Lorek and Sonnenschein (1998) describe how member data of 'weight' and 'age' and member functions of 'catch_mouse', 'growOld' and 'fly' were added to a class definition from a library to create a new class called 'eagle'. Once the programmer has finished customising source code it can then be compiled to create a stand-alone application that can be used to carry out simulation runs.

2.3.4 Formulation of a strategy for model development

The following principles can be abstracted from the material in Section 2.3:

- Formal model designs offer many advantages over informal designs. Formal designs are unambiguous and complete. In addition they can be used to automatically generate runnable models. They may also support other tasks such as model interrogation and model checking.

- Collaborative modelling is very common and is an effective way of addressing many problems. Collaborative modelling is a natural way of handling the adaptation of models to different forest types and different modelling objectives, but requires the adoption of well-defined procedures.

- Common interfaces for models are a way of increasing the effectiveness with which models can be used and exchanged.

- Sharing and reuse of model content can increase the effectiveness with which new models can be developed.

- Sharing and reuse of model content can be facilitated by modularity in which modules are 'tagged' with information concerning their requirements. Other methods that rely on encapsulation are less useful.

3. Requirements specification for SYMFOR

Requirements analysis and specification is recognised as being an important part of the process of software development. Requirements analysis typically involves two stages: identification of *stakeholders* and elicitation from stakeholders of the features they would like to see included in the final software. A stakeholder may be defined as anyone possessing (at least potentially) a strategic interest in the development and use of the software. The concept is more valuable than that of 'users' with respect to requirements specification. This is because often those who have a stake in software such that they would seek to influence its development and benefit from a good design will not be those who will sit down at a computer and interact directly with the software. For example, someone who is responsible for software in a government institution may not themselves ever use software, but may have clear ideas about features that they would like to see incorporated and benefit from the effective use of the software.

3.1 STAKEHOLDERS

The following analysis considers potential stakeholders in SYMFOR. Several different activities formed the basis of the analysis presented in this section:

- Discussions with forestry professionals in Indonesia (Muetzelfeldt and Young, 1996; 1997).
- Feedback from workshops and presentations undertaken in Indonesia (Muetzelfeldt and Young, 1996; 1997).
- Studies undertaken by Indonesian forestry professionals as part of their Studies at University of Edinburgh University (Wibowo, 1995; Raharja, 1996).

3.1.1 Government Researchers

In general this class of stakeholder is responsible for applied research *i.e.* research which yields results of direct application for management. This may take the form of conducting trials of new silvicultural systems *e.g.* experimenting with different harvesting cycles or intensities. Government researchers may advise government on policy for forest exploitation. Government Research departments are unlikely to be able to have enough resourcing to establish and maintain a high density of permanent sample plots over a large geographical area. Instead they may have a role co-ordinating and supporting researchers in concession companies.

Requirements:

- evaluation of silvicultural treatments.- Government researchers may have a role in conducting preliminary investigations of new techniques.

- efficient yield table generation - There is a need for researchers to produce yield tables that can be of use to forest managers.

- creation of new model designs - There is a need for researchers to be able to adapt models to respond to changes in policy with respect to forest utilisation and to respond to new insights or findings in research. In addition to creating models for their own use, government researchers may also create new models based on designs supplied by outside agencies such as concession companies.

- realisation of new model designs – There is a need for researchers to be able to implement new model designs as runnable models.

- full technical documentation of models - Researchers working for concession companies will be responsible for the design of data collection protocols for calibration and initialisation data. For them to do this effectively they must have a clear idea of the structure of the model and the way in which the data is used.

3.1.2 Research and Development personnel in Concession companies

This class of user is responsible for managing and directing applied research in forest concessions. This will typically involve the setting up and maintenance of permanent sample plots within the concession area - this is currently a statutory duty for the concession companies in Indonesia. They also have a role in advising concession managers on the productivity of different silvicultural systems. They may produce research notes on the findings from their research. R&D personnel have identical requirements to Government Researchers, with the possible exception that they may not be able to realise models themselves. Instead they may be able to transmit new designs to the Government researchers, who may undertake the process of implementation and return a runnable model to them.

3.1.3 University Researchers

This class of stakeholder is responsible for theoretical and applied research. Typically they will be less constrained by the need to produce results that are immediately applicable in management. This may enable them to undertake a diverse set of investigations, some of which may ultimately form the basis of more intensive applied research. For example, they may be able to study the effect of simulation plot size on estimation bias caused by edge effects.

Requirements:

- flexible methods for display of output - The range of subjects for investigation by this class of stakeholder will be large.
- meaningful display of model results - When undertaking a large number of investigations the importance of eyeballing data is greater.
- creation of new model designs - The diversity of investigations likely to be undertaken by this class of stakeholder means that this feature is especially vital.

- full technical documentation of models - University researchers are likely to be especially interested in behaviour caused by subtleties in the model design and implementation process (such as order in which model calculations are performed). While often the impact of these factors may be minimal sometimes it can be extremely important.

3.1.4 Forestry Trainers

This class of stakeholder is responsible for training future forest managers. For the subject of silviculture there is a need to cover the advantages and disadvantages of different silvicultural systems and to cover principles such as experimental design and data analysis. While it is possible to analyse the results from pre-existing experiments or to discuss the principles of experimental procedure it is difficult to tackle these subjects in an interactive (and therefore possibly more stimulating and educationally beneficial) manner.

Requirements:

- interactive simulation - Simulations run fast enough so that it is possible for users to interact with them in real time.
- ease of use - Students will not have time to spend becoming acquainted with subtleties of software operation.
- meaningful display of data – The purpose is to stimulate students and good visualisation is very useful in this respect.

3.1.5 Forest Operations Managers

This class of stakeholder is responsible for managing the forest on a day to day basis. Amongst the myriad decisions they take are:

- splitting up of forests into management blocks.
- deciding on the order in which blocks should be harvested.

Forest managers will not need to use simulation software in their everyday work. However to be effective managers they should understand the concepts of precision, accuracy and generality in productivity estimation. This will help them to assess risk.

Requirements:

- software should provide growth and yield tables that can be used in day-to-day management decisions.

Since this class of stakeholder will not have a high need to interact directly with the software they do not have any direct requirements for software design.

3.2 REQUIREMENTS

This section develops and synthesises the requirements of the last section. The presentation of requirements given in this section is 'flat' *i.e.* no attempt is made to exploit hierarchical relationships between different requirements. This is despite the fact that some of the requirements are clearly subsidiary to others. For example, modularity may help support collaborative modelling. The reason that hierarchical classifications are avoided is that a single requirement may be important in relation to more than one higher -level requirement, so that it is difficult to use a single hierarchical classification.

Forest production estimates

1. *The framework should provide long term estimates of forest production.*
   Growth and yield estimates are essential for sustainable or wise forest management (Section 1.2). Second and later cycle timber yields (*i.e.* long term predictions) are important as well as first cycle yields.

2. *The framework should be able to cope with spatial variability in forests.*
   Spatial variability within areas of forest of the same type may be substantial (Section 1.2.2).

3. *The framework should be able to predict the impact of different harvesting practices (such as reduced impact logging) and different harvesting intensities on forest production.*

   There is a growing recognition that the way in which harvesting takes place is important as well as the harvesting intensity in determining future growth and yield (Section 1.2.1).

4. *The framework should be able to predict the impact of silvicultural operations other than harvesting (e.g. replanting or cleaning of the stand) on forest production.*

   Silvicultural operations may, in some circumstances, have a significant affect on growth and yield (Section 1.1).

Representation of forest processes

5. *Models should capture disturbance in forest stands.*

   Disturbance is an important concept for explaining and predicting dynamics in forest ecosystems (Section 2.1.3)

6. *Models should capture recovery from disturbance in forest stands.*

   Recovery from disturbance involves several processes, any one of which may constrain recovery and so affect the long-term dynamics of the forest (Section 2.1.3).

7. *Models should capture species-specific behaviour as far as possible.*

   Species may vary substantially in their response to environmental conditions, and this variability can have important consequences for forest dynamics (Section 2.1.3).

8. *Models should capture local interactions and environment within the stand.*

   Processes of disturbance (*e.g.* tree fall) and recovery from disturbance (*e.g.* competition between trees) are essentially local phenomena. Similarly, silvicultural processes *e.g.* design of skidtrails are often undertaken with reference to local conditions within the stand (Section 2.1.3).

Data collection and handling

6. *Models in the framework should utilise permanent sample plot data where possible*
Permanent Sample Plot data is widely available, standard techniques exist for their collection and the resourcing required to establish and maintain the plots is not prohibitive.

7. *Data requirements for calibration of models should not be excessive.*
It is expensive to augment PSP data and representation of extra processes may not necessarily improve the quality of model predictions (Section 2.3.2).

8. *Recalibration of models should be minimised.*
Recalibration is time-consuming. It is also expensive, in that it may involve the collection of new data sets.

Framework architecture

9. *The framework should support collaborative modelling.*
Collaborative modelling is both common and desirable (Section 2.3.1).

10. *The framework should support creation of new model designs.*
There is nearly always a need to redesign models to cope with new circumstances (Section 2.3.2).

11. *The framework should support sharing and reuse of model content.*
Specific provision for sharing and reuse of model content is one way in which both collaborative modelling and the creation of new model designs can be supported (Section 2.3.3).

Modelling language

12. *The framework should have a formal modelling language capable of specifying the content of model designs.*

    The framework must be capable of creating and handling different model designs. To do this it requires a formalism for capturing details of the content of models. A formal language can be engineered to contain sufficient semantic and syntax to capture content details.

13. *The language should make use of concepts and terms from System Dynamics where possible.*

    Systems Dynamics is a well-established discipline with its own terminology. Using Systems Dynamics terminology may make it easier for users with experience of Systems Dynamics to operate the framework and may also make it easier for users to relate SYMFOR models to the modelling literature (Section 2.2.6).

14. *The language should be able to specify grouping of logically related design elements.*

    Sometimes sets of design elements (*e.g.* model variables) naturally belong together. For example, the variables used to describe a representation unit such as an individual tree naturally belong together. It is more efficient to handle these variables as a group for some activities (Section 2.2.6).

15. *The language should be able to specify sharing of design elements by representation units.*

    In disaggregated stand representations the stand is represented by a set of representation units. These units all share features such as model variables. For example, a set of trees may be represented by stem-diameter, height and crown-point in some model. The language should be able to exploit this so that representation units need not be individually specified (Section 2.2.6).

16. *The language should be able to specify creation and destruction of representation units.*

    This is a requisite for some kinds of model (those using individual-based representations or cohort representations of trees) (Section 2.2.6).

User interaction

17. *There should be a common interface for all models within the framework.*
The learning curve for interacting with new models is substantially reduced if all models have a common interface (Section 2.3.3).

18. *The framework should be usable by people with a limited technical background.*
It is desirable that a large constituency of users, and not simply the original programmer will use the software. In particular, users should not need to have a programming background to be able to interact with models in the system.

19. *The framework should support interactive simulation.*
Interactive simulation is an important way of making the software more effective for teaching and training applications.

20. *The framework should provide methods for display of results that can be used with models possessing different content.*
It is important to supply users with meaningful information (as opposed to large output files of results) because it helps stimulate user interaction and helps users to uncover pathological model behaviour. Flexibility is essential for coping with different classes of user identified in Section 3.1 (*e.g.* research scientists and forestry trainers) and with models which differ in content.

21. *The framework should provide information on each model in the system that is meaningful, unambiguous and complete.*
Section 2.3.1 discusses how some forms of model design can be incomplete or ambiguous. There is a need to provide unambiguous and complete information on models in the system.

4. Design of SYMFOR ontology

Design as it used here refers to the conceptual phase of development that spans the gap between the Requirements phase and the Implementation phase. Design in SYMFOR is composed of SYMFOR ontology design and SYMFOR software design. Ontology design here refers to the process of refining concepts and defining terms used in SYMFOR. Software design involves deciding on how software can be developed so as to meet the given requirements.

This chapter follows a similar format to that of a scientific paper. An introductory section briefly describes ontologies and their uses; a methods section details how the SYMFOR ontologies were developed; an ontology description section contains the informal ontology; a discussion section evaluates the methodologies used and compares the SYMFOR ontology with other modelling ontologies.

4.1 INTRODUCTION TO ONTOLOGIES

Many different modelling systems are currently in use (Section 2.3). As discussed in Section 2.3.1 model designs draw on concepts such as state variable, intermediate variable, parameter *etc*. Model designs produced using different modelling systems may use subtly different conceptualisations. For example, in the ModelMaker modelling system (Cherwell Scientific, 1998) it is possible to specify a net rate of change for 'compartments' without attaching any 'flows'. In other modelling systems such as Stella, (Systems Thinking Experimental Learning Laboratory, 1994) 'compartments' can only change by increment or decrement associated with attached 'flows'. Variation in conceptualisations across modelling systems creates two problems:

- communication of the conceptualisations to other modellers so that they can understand them and/or use them for themselves;
- translation of model designs so that they can work in different modelling systems.

Both of these problems may be addressed by the development of *ontologies* for different modelling systems. Ontologies are specifications of the conceptualisations used in a particular area of interest (Gruber, 1993). They aid communication of conceptualisations in that they can serve as reference material for other modellers.

Ontologies assist with the task of translation because their development can make it easier to relate the conceptualisations used in one system to the conceptualisations used in a different system. An understanding of these relationships is a pre-requisite for the production of translation algorithms for converting a model from one system to another. Concepts in different ontologies may be related in a number of different ways, one of the most important being *inclusion* (Farquhar *et al.*, 1995).

Inclusion is a process in which an ontology uses concepts from another ontology without modification. Inclusion is useful because it avoids duplication of effort and also because it helps expose and consolidate similarities between different ontologies, so aiding the development of translation algorithms. For example, consider a case in which two modelling ontologies both use some concept of 'natural number'. When the two are developed separately without any inclusion of pre-existing ontologies then each must include its own definition of 'natural number'. This is a problem because 'natural number' may be differently defined: some mathematicians assume zero is a natural number, while others exclude zero. It is therefore possible for the two modelling systems to have different definitions. If the definitions are expressed using different terminology or concepts then it is difficult to compare the two.

When an ontology for 'number systems' is included in both systems then it is easier to assess the compatibility of modelling systems. Even if it is desirable to use different concepts from those of the included ontology, this can be achieved and, what is more, explicitly exposed in the new ontology. For example, it the 'number system' ontology might define 'natural numbers' to include zero, but it may be required to refer to the set of natural numbers with zero included in the new ontology. This can be achieved by defining a new concept *e.g.* 'natural numbers2' might be defined as 'the set of all "natural numbers" aside from zero'. This concisely exposes the relationship between the conceptualisations used in the different ontologies.

Ontologies may be particularly effective in assisting with translation when combined with the development of an *interchange* format for models. This is a model format engineered so that it can capture details of model designs constructed using a wide variety of different modelling systems. The interchange format is useful because it means that an individual modelling system need only perform one translation for model designs to become available for use in many different systems.

There are many different kinds of ontology (Uschold and Gruninger, 1996). It is however possible to recognise two broad groups:

- informal – these are ontologies expressed in natural language. These ontologies are usually concise and comprehensible, but may be ambiguous or incomplete.

- formal – these are ontologies expressed in artificial, formally defined languages. Languages based on first- order predicate logic have been shown to be suitable for capturing details of many conceptualisations (Gruber, 1993). First-order predicate logic is a powerful representation formalism, and can accommodate other kinds of representation such as frame or object based representations or semantic networks (Muetzelfeldt *et al*, 1989). These ontologies are usually more complete, and less ambiguous than informal ontologies, but may be difficult for domain workers to understand.

Ontologies have been developed and used for diverse applications such as medical informatics (Gennari *et al.,* 1995), engineering (Gruber and Olson, 1994) accounting and the modelling of activities carried out in businesses (Stader, 1996). However, as yet there has not been an attempt to develop an ontology specifically for ecological modelling.

Several tools exist for the construction of ontologies. These tools are useful because they may provide easy-to-use graphical user interfaces for ontology construction and also may provide functionality for specialised activities such as automatic inclusion of existing ontologies. Ontolingua is one such tool (Gruber, 1993). It has support for inclusion of existing ontologies and it supports collaborative ontology development.

There are three reasons for using an ontology to describe the conceptualisations developed for model design in SYMFOR:

- There is a need to provide a rigorous account of the modelling concepts used in SYMFOR as some of the requirements for SYMFOR listed in Chapter 3 are met by adopting particular concepts.

- There is a need to assess the suitability of currently available tools for constructing ontologies for ecological modelling.

4.2 METHODS

Two representations of the SYMFOR ontology were created: an informal representation and a formal representation. This follows the practice of Uschold *et al.* (1998). The two representations possess complementary characteristics. The informal ontology is written in natural language and should be comprehensible by workers in the domain of ecological modelling. The formal representation is more technically precise and conforms to a standard (the Ontolingua standard, Gruber (1993)) that allows sharing and reuse of ontologies by different software agents and contributes to shared understanding of concepts within a community of ontology users.

4.2.1 Creation of informal ontology

The informal representation was developed to correspond to the formal representation of the ontology. Uschold *et al.* used three concepts to express details of their ontology that are also used in this chapter. The three are:

Entity - this is a thing *e.g.* a modelled tree, a module, a parameter

Relationship - this is a way in which two Entities can be associated *e.g.* a tree may be related to an attribute by the Relationship 'has-attribute', a parameter may be related to a module by the relationship 'has-parameter'

Activity - this is something which takes place over time *e.g.* an evaluation of tree stem-diameter

In the informal version of the ontology highly technical language was avoided. This involved replacing some of the very precise language of the formal ontology with less precise language that was more comprehensible. In many cases the context of a phrase should allow its meaning to be correctly inferred. Three main simplifications were used:

- No distinction was made between the various kinds of 'is-a' relationship. There are at least three different kinds of 'is-a' relationship: two kinds of 'instance-of' (instances share the same slots as the class and instances share same values as the class) and 'is a subclass of'. For example, a 'state variable is an initialised datum' indicates that 'state variable' is a subclass of 'initialised datum' while 'stem diameter is a state variable' indicates that 'stem-diameter' is an instance of 'state variable'.

- No distinction was made between classes and slots in some cases. For example, 'state variables are updated...' is more correctly written as 'values of state variables are updated'.

Two conventions are adopted in this chapter. Terms defined in the ontology are written with capitals throughout, terms from the meta-ontology begin with a capital letter. The informal ontology is described in Section 4.3.

4.2.2 Creation of the formal ontology

The formal ontology was created using the Ontolingua ontology development environment. The environment is described in Gruber (1993), Fikes and Farquhar (1997). This tool has a number of special features:

- high level of support for ontology representation. Ontology representation is achieved using the *frame ontology* supplemented where necessary by user-defined Ontolingua *axioms*.

  In the frame ontology a *class* is a logical grouping of *slots*. Each slot may have one or more values which indicate the characteristics of an individual of the class. For example, in the SYMFOR ontology there is a class called SCALAR PARAMETER. This class has slots for 'Name', 'Minimum Value', 'Maximum value', 'Default', 'Units' and 'Description'.

  Slots have several *facets* that constrain the way that they can be used. *Slot-value-type* is a facet that determines the classes that can be accommodated in the slot. For example, the slot-value-type of the Minimum Value slot of a SCALAR PARAMETER IS 'REAL NUMBER. S*lot-cardinality* is a facet which represents the number of values that a slot can accommodate. For example, the class MODULE has a slot called 'has-parameter'. In this case the slot cardinality will be 'zero or greater'.

Classes can *inherit* slots from other classes. A class receives slots from another class if it is defined to be a subclass of the other class. For example, in SYMFOR the class EVALUATED ATTRIBUTE has a slot called 'has-module'. Five classes in the ontology are subclasses of EVALUATED ATTRIBUTE and all of them will possess the slot 'has-module'. The inheritance is transitive, so that subclasses of a class that is itself a subclass will inherit slots of the top-level class.

Axioms are used when the nature of a concept in the ontology is not fully expressible using frame-language concepts such as classes and slots. For example 'sibling' may be defined as a slot on a 'person' class. The slot-value-type is 'person' and the cardinality is 'zero or greater'. However, to fully capture the nature of the relationship an extra constraint is required, *i.e.* that the relationship can only exist between two persons if the two have a shared parent. In Ontolingua, constraints of this kind can be expressed using first order predicate logic. The particular language used is Knowledge Interchange Format (KIF) and is discussed below.

- functionality for ontology presentation. In Ontolingua ontologies are presented in an *object-based* manner. By this is meant that the organisation of ontology content on the screen that users see when using the environment reflects logical groupings of concepts (such as classes) defined in the ontology. Hypertext links allow users to navigate through the ontology. A special edit mode allows users to modify details of the ontology or to create new classes and slots.

- support for *inclusion* of existing ontologies. Inclusion was discussed in Section 4.1. Ontolingua allows users to include pre-existing ontologies when constructing a new ontology. Inclusion is non-trivial because of the scope for inconsistencies between ontologies. For example, a symbol such as 'parameter' may be defined in the new ontology and in an included ontology. Ontolingua contains special functionality for dealing with such inconsistencies.

- ontology access via World Wide Web. The Ontolingua Ontology Editor and existing ontologies are accessible via the World Wide Web (Farquhar *et al.*, 1995). This means that it is possible for groups of users to collaborate on ontology creation. It also means that it is easier to make use of previously created ontologies.

- support for translation of ontologies. Ontolingua has functionality for translation of ontologies into different formats. For example Clips, Epikit, and Prolog representations can be generated from an Ontolingua representation of an ontology (Gruber, 1993). This is useful as different organisations may need to use the same ontology, but may commit to different representational formalisms for historical or other reasons.

The formal version of the SYMFOR ontology was created using Ontolingua. Production of a formal ontology using Ontolingua ostensibly involves two steps: specification of classes and slots in the ontology and specification of axioms required for the ontology. However, most of the important work in ontology creation will take place before or during the first stage *i.e.* there will have been a thorough analysis of the problem which the ontology will address and the development of suitable concepts for the ontology. The second stage, that of encoding axioms contrasts with the first in three ways. First, it is technically involved (it involves the use of a special formal language based on first-order predicate logic called Knowledge Interchange Format (KIF)). Second, as most of the work on concept development has already taken place, it is more mechanical in nature than the first step. Third the encoding involved in the second step is less important than that of the first step as often the nature of the axiom can be satisfactorily indicated using informal language (consider the case of the 'sibling' relationship described above). For these reasons only the first step of Ontolingua ontology development was undertaken and natural language descriptions of the axioms are given .

4.3 DESCRIPTION OF THE INFORMAL ONTOLOGY

The informal ontology makes use of diagrams and (relatively) informal language to de fine the concepts used in SYMFOR to build models. There are three parts to the description of the informal ontology. The Ontology diagrams are semantic networks that show two kinds of relationship between terms in the ontology (Section 4.3.1). The Overview is a text description of the entire ontology (Section 4.3.2). This is followed by a more detailed definition of each of the terms that should allow them to be used appropriately by users of the ontology (Section 4.3.3).

4.3.1 Ontology diagrams

The Ontology diagrams indicate when certain kinds of relationship hold between terms in the ontology. In the SYMFOR ontology all terms are classes that can participate in two kinds of relationship:

- Inheritance relationships. Classes may inherit certain properties from other classes. For example, a 'dog' class may inherit properties from a 'mammal' class. This means that the 'dog' class possesses all properties of the 'mammal' class (but may possess other properties as well). A class gaining properties from another class is a *subclass* of the other class. These kinds of relationships between terms of the SYMFOR ontology are shown in Figure 4.1

- Association relationships. Classes may be related such that instances of one class can be associated with instances of another class. For example an instance of a 'dog' class may be associated with one or more instances of a 'flea' class. These kinds of relationships between terms of the SYMFOR ontology are shown in Figure 4.2.

Figure 4.1: Diagram showing 'subclass-of' relationships in the SYMFOR ontology. Each arrow in the diagram indicates that the thing at the origin of the arrow is a subclass of the thing at the end of the arrow.

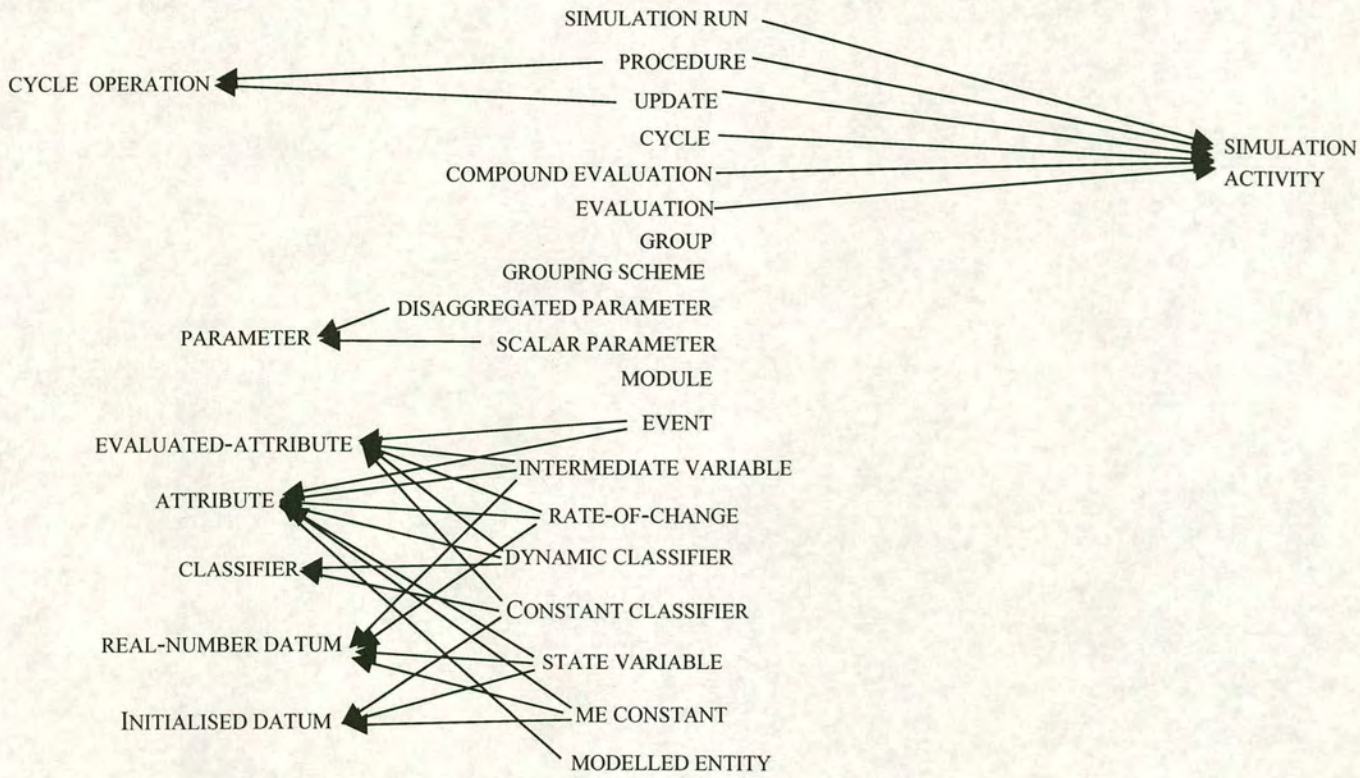
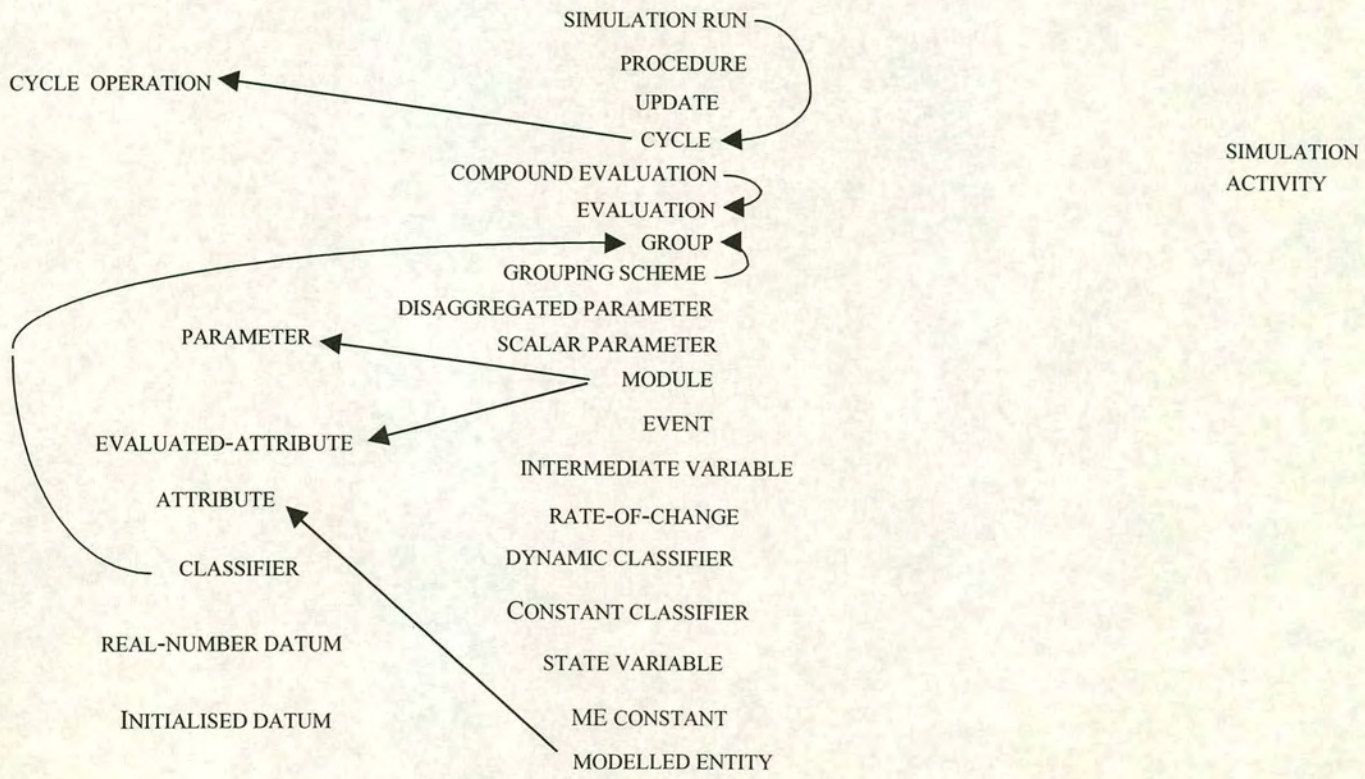


Figure 4.2: Diagram showing 'has-a' relationships in the SYMFOR ontology. Each arrow in the diagram indicates that the thing at the origin of the arrow is associated with one or more instances of the things at the end of the arrow.



4.3.1 Overview of the SYMFOR Ontology

The forest stand is represented by a number of MODELLED ENTITIES (MEs) in SYMFOR simulations. Each ME possesses a number of ATTRIBUTES.

There are seven different kinds of ATTRIBUTE: ME CONSTANTS, STATE VARIABLES, INTERMEDIATE VARIABLES, CONSTANT CLASSIFIERS, DYNAMIC CLASSIFIERS, RATES- OF-CHANGE and EVENTS. The specialisation of these kinds of ATTRIBUTE is shown in Figure 4.1. Those kinds of ATTRIBUTE that are EVALUATED ATTRIBUTES are subject to regular EVALUATIONS by algorithms contained in MODULES. Each MODULE may have one or more PARAMETERS. There are two kinds of PARAMETER: SCALAR PARAMETERS and DISAGGREGATED PARAMETERS. DISAGGREGATED PARAMETERS are associated with one or more GROUPING SHEMES. GROUPING SCHEMES contain two or more individual GROUPS.

A SIMULATION RUN is formed by the repetition of one or more SIMULATION CYCLES. A SIMULATION CYCLE has one or more SIMULATION OPERATIONS. There are two kinds of SIMULATION OPERATION: PROCEDURES and UPDATES. Each PROCEDURE consists of one or more COMPOUND EVALUATIONS. COMPOUND EVALUATIONS consist of a set of EVALUATIONS.

4.3.2 Definition of terms in the SYMFOR ontology

4.3.2.1 MODEL ENTITIES AND ATTRIBUTES

These terms are primarily used to specify a certain kind of data that are used in the representation of forest stands in SYMFOR models. The data describe the state of the model, and any model characteristics that depend upon the state.

MODELLED ENTITY

This is an Entity that corresponds to something in the real world and is represented in a simulation.

Examples:

- an individual tree;
- an individual gridsquare.

Notes:

MEs are instances rather than classes.

1. The use of the term 'object' is avoided. This is primarily because it is often connoted with concepts such as inheritance, encapsulation or polymorphism and no such concepts are present in SYMFOR.

HAS-ATTRIBUTE

This is a Relationship between an ME and an ATTRIBUTE.

ATTRIBUTE

ATTRIBUTES specify representation data used for MEs in a simulation. There are eight different kinds: ME CONSTANT, STATE VARIABLE, CONSTANT CLASSIFIER, DYNAMIC CLASSIFIER, INTERMEDIATE VARIABLE, RATE-OF-CHANGE and EVENT.

Examples:

- tree stem diameter;
- gridsquare organic matter;
- tree mortality.

Notes:

1. Informally an attribute can be thought of as 'something about an ME that is explicitly captured'.
2. ATTRIBUTE is quite similar to the concept of 'attribute' used in other systems. However, 'attribute' is often synonymous with 'item of data'. In SYMFOR ATTRIBUTES are also EVENTS such as 'mortality'.

CLASSIFIER DATUM

ATTRIBUTES that are CLASSIFIER DATUMS identify which GROUP from a GROUPING SCHEME an individual ME belongs to. There are two kinds of CLASSIFIER DATUM: CONSTANT CLASSIFIER and DYNAMIC CLASSIFIER.

Examples:

- tree species group;
- tree size class.

Notes:

1. In other systems the idea of a thing belonging to a set of things is captured using a Relationship such as 'is-a-member-of'. This Relationship does not logically belong to either the thing or the set. This differs to SYMFOR where the 'Relationship' effectively belongs to the thing *i.e.* set membership is denoted by a CLASSIFIER DATUM which is an attribute associated with a particular ME.

CLASSIFIER GROUPING-SCHEME

This is a Relationship between a CLASSIFIER DATUM and a GROUPING SCHEME whereby values associated with the CLASSIFIER DATUM identify one GROUP from the GROUPING SCHEME.

Examples:

- tree species-group is associated with species-group;
- cohort species-group is associated with species-group.

REAL-NUMBER-DATUM

ATTRIBUTES that are REAL-NUMBER DATUMS capture something about an ME that can be expressed using a real number value. There are four different kinds of REAL NUMBER DATUM: ME CONSTANT, STATE VARIABLE, RATE-OF-CHANGE, INTERMEDIATE VARIABLE.

Examples:

- tree stem diameter;
- gridsquare organic matter;

INITIALISED-DATUM

ATTRIBUTES that are INITIALISED DATUMS must be assigned a new value when a new ME which possesses the ATTRIBUTE is created. There are three different kinds of INITIALISED DATUM: STATE VARIABLE, ME CONSTANT and CONSTANT CLASSIFIER.

Examples:

- tree stem diameter;
- tree species-group;

EVALUATED-ATTRIBUTE

An ATTRIBUTE that is an EVALUATED ATTRIBUTE must be evaluated regularly in a simulation. By this is meant that the MODULE algorithm associated with the EVALUATED ATTRIBUTE by the Relation HAS-MODULE must be executed each time an EVALUATION of the ATTRIBUTE occurs in a simulation. In the case of EVALUATED ATTRIBUTES other than those that are EVENTS this algorithm assigns a new value to the ATTRIBUTE. There are four kinds of EVALUATED-ATTRIBUTE: DYNAMIC CLASSIFIER, RATE-OF-CHANGE, INTERMEDIATE VARIABLE and EVENT.

Examples:

- tree height (when this is recalculated on the basis of diameter each year)

ME CONSTANT

This is a kind of CAPTURED DETAIL that is a REAL NUMBER DATUM and an INITIALISED DATUM. The value of an ME CONSTANT does not change for the lifetime of an ME (as it is not an EVALUATED ATTRIBUTE).

Examples:

- x and y coordinates of trees.

Notes:

In some respects ME CONSTANT is similar to a STATE VARIABLE as it is associated with individual MEs and is required to specify the state of the stand at a point in simulation time.

In some respects ME CONSTANT is similar to the traditional conception of 'parameter' in that it is used in model calculations and its value(s) are not affected by model processing. However, PARAMETERS in SYMFOR are by definition values not primarily associated with individual MES. This conceptualisation of PARAMETER is more natural because gathering values for things treated as ME CONSTANTS (*e.g.* x and y co-ordinates of trees) is logically similar to gathering initial values for state variables and dissimilar from parameter estimation procedures such as regression analysis.

STATE VARIABLE

- This is a kind of ATTRIBUTE that is a REAL NUMBER DATUM and an INITIALISED DATUM.

Examples:

- tree stem diameter.

Notes:

1. STATE VARIABLES change through addition or subtraction of RATES-OF-CHANGE in UPDATES.
2. STATE VARIABLES are synonymous with 'compartments' from compartment-flow modelling and 'stocks' or 'levels' from system dynamics (Haefner, 1996).

CONSTANT CLASSIFIER

This is a kind of ATTRIBUTE that is an INITIALISED DATUM and a CLASSIFIER DATUM.

Examples:

- tree species group identity.

Notes:

1. A CONSTANT CLASSIFIER is used when an ME will not change the GROUP from a GROUPING SCHEME to which it belongs.

DYNAMIC CLASSIFIER

This is kind of CAPTURED DETAIL that is a CLASSIFIER DATUM and an EVALUATED ATTRIBUTE.

Examples:

- tree size-class;

Notes:

1. A DYNAMIC CLASSIFIER is used when an ME may change the GROUP from a GROUPING SCHEME to which it belongs for the lifetime of the ME.

INTERMEDIATE VARIABLE

This is a kind of ATTRIBUTE that is a REAL-NUMBER DATUM and an EVALUATED ATTRIBUTE.

Examples:

- tree crown radius;
- gridsquare shading.

Notes:

1. INTERMEDIATE VARIABLES are similar to the concept of the same name used in compartment flow modelling. They are also known as 'auxiliary' variables (Bossel, 1994; Haefner, 1996).

RATE-OF-CHANGE

This is kind of ATTRIBUTE that is a REAL NUMBER DATUM and an EVALUATED ATTRIBUTE.

Examples:

- tree diameter increment;
- seedling mortality.

Notes:

1. RATES OF CHANGE are very similar to the concept of 'flows' from compartment flow modelling (Haefner, 1996).
2. RATES OF CHANGE are used to increment or decrement the value of STATE VARIABLES in UPDATES.
3. More than one RATE-OF-CHANGE can be applied to the same STATE VARIABLE. The actual change to the value of the STATE VARIABLE in an UPDATE is given by the sum of all the RATES-OF-CHANGE that apply.

EVENT

This is a kind of ATTRIBUTE that is an EVALUATED ATTRIBUTE.

- Examples:
- tree mortality;
- stand logging;

Notes:

1. Informally an event can be thought of as 'something that can happen to an ME.

EVENTS are different from all other ATTRIBUTES in that they do not specify data that must be used in representation of MES. They are treated as ATTRIBUTES because it is useful to treat them as EVALUATED ATTRIBUTES *i.e.* subject to regular EVALUATIONS such that the algorithm in a MODULE is called once for each ME possessing the EVENT.

2. EVENTS do not necessarily result in the destruction or creation of MEs, though they often do.
3. A single EVENT can comprise both entity creation and destruction. This is useful as many creations and destructions are logically associated *e.g.* treefall creation is logically associated with tree destruction.
4. An EVENT associated with an ME can result in the creation or destruction of other MEs *i.e.* its effects are not confined to the originating entity. This is useful for capturing phenomena such as natural disturbance, where a single tree can fall over but can knock down many of the surrounding trees as it falls.

ME TYPE

This is a set of MEs that possess the same set of ATTRIBUTES.

Examples:

- trees in a stand;
- gridsquares in a stand.

Notes:

1. 'set of attributes' refers to properties rather than values in this case i.e. trees are all entities that possess the property dbh rather than all entities which possess a dbh of 10.4 cm.

4.3.2.2 MODULES, PARAMETERS AND GROUPS

These terms are used to specify algorithms used in SYMFOR models. The algorithms are used to determine values for EVALUATED ATTRIBUTES of MEs. Data used by algorithms (other than ME data) are called PARAMETERS.

MODULE

This is an algorithm that is associated with all ATTRIBUTES that are EVALUATED ATTRIBUTES. The MODULE is executed in an EVALUATION.

- Examples:
- the MODULE height2 (an algorithm that uses a quadratic equation to determine the height of a tree based on its diameter).
- the MODULE disturbance4 (an algorithm that assumes each tree in a stand has the same probability of initiating natural disturbance).

Notes:

1. This definition is specific to SYMFOR - there is no concensus within ecological modelling on a single meaning of the term 'module'

HAS REQUIREMENT FOR

This is a Relationship between a MODULE and an attribute whereby the ATTRIBUTE must be present for the MODULE to function correctly.

Examples:

- the MODULE height2 (see above) HAS REQUIREMENT FOR tree diameter.

Notes:

1. ATTRIBUTES that feature in the Relationship may be informally described as 'inputs' for the MODULE.
2. More than one ATTRIBUTE can be in the Relationship for a single MODULE.

HAS PARAMETER

This is a Relationship between MODULES and PARAMETERS whereby the PARAMETER is a value or set of values used in EVALUATIONS of the MODULE.

Notes:

1. A single MODULE can be in the relationship with more than one PARAMETER.

GROUP

This is a 'bin' into which individual MEs can be placed. The GROUP or GROUPS in which a ME can be placed determine the values from those associated with a DISAGGREGATED PARAMETER that are used in EVALUATIONS of ATTRIBUTES associated with the ME.

Example:

- trees can be placed into a GROUP that contains trees between 10 and 30 cm stem diameter.

GROUPING SCHEME

This is a set of GROUPS that are logically related.

Examples:

- tree size classes;
- tree species-groups.

Notes:

Individual MEs can only belong to one GROUP in a particular GROUPING SCHEME.

PARAMETER

A PARAMETER is a value or set of values used in particular way in an EVALUATION. There are two kinds of parameter: SCALAR PARAMETER and DISAGGREGATED PARAMETER.

Examples:

- tree growth rate coefficient;
- stand logging year.

SCALAR PARAMETER

This is a kind of PARAMETER that is associated with one value only.

Examples:

- stand logging year;
- skidtrail width.

DISAGGREGATED PARAMETER

This is a kind of PARAMETER that is associated with one or more GROUPING SCHEMEs. This kind of PARAMETER has a number of PARAMETER values. Each value corresponds to a GROUP (or combination of GROUPS) in the GROUPING SCHEMES.

Examples:

- **c**, a species specific growth rate coefficient. There is one value of **c** for every species GROUP in the species GROUPING SCHEME.

Notes:

1. When a PARAMETER is associated with one GROUPING SCHEME then the set of values associated with the PARAMETER is one-dimensional and there is one value in the set for each GROUP in the GROUPING SCHEME.
2. When the DISAGGREGATED PARAMETER is associated with two GROUPING SCHEMES then the set of values associated with the PARAMETER is two-dimensional. In this case there is one value for each possible combination of GROUPS from the different GROUPING SCHEMES (where each combination consists of one GROUP from each GROUPING SCHEME).

4.3.2.3 Activities

These terms are used to specify details of the operations that take place in model simulations.

EVALUATION

This is a kind of Activity in which an ATTRIBUTE associated with an ME is evaluated *i.e.* the MODULE algorithm associated with the ATTRIBUTE is executed once.

Examples:

- calculation of tree height for an individual tree;
- determination of whether an individual tree will initiate a natural disturbance event.

COMPOUND EVALUATION

This is a kind of Activity which consists of a series of EVALUATIONS of an attribute associated with MEs of a particular ME TYPE. An EVALUATION is performed for each ME of the ME TYPE in turn.

Examples:

- calculation of tree height for all trees;
- determination of whether individual trees will initiate a natural disturbance event for all trees.

PROCEDURE

This is a kind of MODEL OPERATION that performs one or more COMPOUND EVALUATIONS in turn.

Examples:

- A PROCEDURE called growth which performs COMPOUND EVALUATIONS of tree height, shadeindex and diameter increment.

Notes:

1. PROCEDURES are used to group and set the order of those COMPOUND EVALUATIONS which have a logical sequence. For example, it is clearly desirable to calculate tree height before calculating tree shading index if the former affects the latter. The reverse order will never be appropriate. However, when considering the position of a COMPOUND EVALUATION such as natural disturbance which results in the destruction of trees it is not clear whether it should be before, after or in between the other two COMPOUND EVALUATIONS. For this reason it should be placed in a different PROCEDURE and manipulated independently.

SIMULATION OPERATION

This is the union of PROCEDURE and UPDATE.

Examples:

- growth procedure;
- annual update.

CYCLE

This is a sequence of MODEL OPERATIONS that is repeatedly performed in a SIMULATION RUN and that corresponds to a particular time period.

Examples:

- annual cycle (corresponds to a year);
- monthly cycle (corresponds to a month).

Notes:

Cycles are similar to the concept of 'iterations' used in numerical analysis. However, iterations are usually assumed to be atomic

PERFORMS OPERATION

This is a Relationship between a CYCLE and a SIMULATION OPERATION whereby the CYCLE performs the SIMULATION OPERATION once when it is executed.

Examples:

- annual cycle may perform growth procedure;
- monthly cycle may perform seedling mortality procedure.

Notes:

Each CYCLE can perform more than one SIMULATION OPERATION.

UPDATE

This is a MODEL OPERATION in which the values of RATES-OF-CHANGE are added to STATE VARIABLES and the creation or destruction of MEs is fully realised.

Examples:

- annual update;
- monthly update.

Notes:

Creations and destructions are initially specified in EVALUATIONS . However newly created or destroyed MEs are by default distinguished from the others until the time of the next UPDATE. This is done to allow users to specify that new creations and destructions are ignored in EVALUATIONS that occur between UPDATES so that the order in which EVALUATIONS are carried out does not influence their outcome. (If the user would prefer that creations and destructions are immediately taken account of they can schedule an UPDATE immediately after a series of EVALUATIONS or they can use MODULE algorithms for EVALUATIONS which specifically take account of newly created or destroyed entities.)

1. Each CYCLE has a unique UPDATE.

SIMULATION RUN

This consists of a CYCLE repeated an arbitrary number of times.

4.4 DISCUSSION

4.4.1 Comparison of SYMFOR ontology with other modelling ontologies

SYMFOR differs from object-oriented systems in that two different kinds of object are distinguished: MEs and GROUPS. MEs and their data describe the state of a modelled system at a point in time. The number of MEs can change in the course of a run. They are assigned values when a model run is initiated. These values are changed in a series of iterations in the course of a simulation. 'Tree' is an example of an ME.

GROUPS are used to store PARAMETER values used in the model. The number of GROUPS never changes in the course of a simulation. They are assigned values when PARAMETER files are read in or when parameter values are edited. 'Species' is an example of a GROUP (in an individual-based representation). For example, 'species' could be used to store species-specific growth coefficients and species-specific allometric coefficients.

The distinction was used because forest scientists and modellers naturally make it. The strategy of separating stand structure data from other input data is common in forest simulation models. For example, Alder (1995) describes a size-class based model called GHAFOSIM. The model accepts two input files: a stand table and a file containing size-class transition probabilities.

One of the central concepts in system dynamics is that of 'state'. Modellers expect to have to initialise the state at the start of a simulation run. This typically involves reading a data file that contains information on the initial values of state variables. STATE VARIABLEs are associated with MEs such as trees but not with GROUPS such as species. They similarly also expect to have to input parameter values. If the distinction between MEs and GROUPS is not made then it is difficult to structure software to reflect the natural distinction. For example, in SYMFOR reading parameter values and initialising the stand state are handled as separate activities. This would be difficult if not impossible if MEs were not distinguished from GROUPS.

The second feature of note is the way in which the term 'parameter' is used. In the classical Systems Dynamics notation of Forester (Haefner, 1996) a 'parameter' refers to a value used in a model that is not influenced by other variables in the model. In SYMFOR this applies to two entities: ME CONSTANTS and PARAMETERS. The two are distinguished because they are used in different ways in a simulation. Table 4.1 lists the differences between the two concepts.

Table 4.1: Comparison of ME CONSTANTS and PARAMETERS used in SYMFOR

ME CONSTANTS	PARAMETERS
Value assigned when ME initialised.	Value assigned when parameters are read from file.
Value associated with ME such as 'tree'.	Value associated with GROUP such as 'species'.
Number of values may change in the course of a simulation (when MEs are created or destroyed).	Number of values fixed.

The third feature of note in the ontology is the definition of MODULE. This is taken to refer to an individual algorithm that undertakes the evaluation of an attribute. Other schemes define modules quite differently such as templates for MEs or as submodels (Section 2.3.3).

The final feature of interest in the ontology is that it facilitates the explicit sequencing of model calculations. This is useful because there is no logical order for some model calculations. For example, it is not obvious whether, growth, or recruitment should be calculated first in a model iteration. This is important because the order in which calculations take place can influence the results obtained (Bugmann *et al.*, 1996). An arbitrary rule for deciding the order of calculations is therefore inappropriate. The ontology must therefore be able to capture details of model sequencing.

4.4.2 Use of Ontolingua for Ontology capture

There were several advantages of using Ontolingua. One of the most important was the use of the frame ontology. Concepts in this ontology (*i.e.* classes, slots *etc*) are extremely intuitive and it is quite easy to capture much of the detail of the SYMFOR ontology using these concepts. In addition, the way that the ontology is presented in Ontolingua makes it very easy to specify classes and slots and to browse the ontology.

One way in which the ontology could be improved is by the creation of KIF statements to capture details of the SYMFOR ontology that cannot be expressed using solely frame language concepts. This, if it had been undertaken, would have involved the creation of KIF statements (Gruber, 1993). While KIF is very powerful and expressive, it is also very terse. In addition as it is based on first order predicate logic a knowledge of the latter is required before it can be used. The axioms required in the case of the SYMFOR ontology are:

- Relationship between MODULES and EVALUATED ATTRIBUTES should be constrained. Not every MODULE is capable of being used to evaluate an individual EVALUATED ATTRIBUTE *i.e.* MODULES are specific to particular EVALUATED ATTRIBUTES. For example, a MODULE for calculating tree crown point cannot be used to evaluate tree height. (If the MODULES were simply equations then it might be possible to interchange them).

- Relationship between COMPOUND EVALUATION and EVALUATION should be constrained. A COMPOUND EVALUATION is responsible for evaluating each instance of a particular ATTRIBUTE. For example, a COMPOUND EVALUATION may specify that tree height will be evaluated for all trees. The same evaluation cannot evaluate more than one ATTRIBUTE *e.g.* tree height and tree crown-radius. In addition it must evaluate all instances of an attribute.

Relationship between CLASSIFIERS and GROUPING SCHEMES should be constrained. A CLASSIFIER must be associated with one GROUP from a GROUPING SCHEME and each ME possessing the CLASSIFIER must be associated with exactly one GROUP from the GROUPING SCHEME. For example, the CLASSIFIER 'species' associated with tree MEs can only be associated with one 'species' GROUP from the 'species' -GROUPING SCHEME.

A new axiom defining the relationship that exists between MEs and ME TYPES is required.

The frame language supplies two concepts that are used to specify that slots possessed by an entity are partially specified by a related class: 'Sub-class' and 'Individual'. Both of these are insufficient for capturing the required relationship because they are used to specify slots rather than slot values. Specification of the latter is required of the relationship between ME types and individual MEs. For example, a 'tree' ME TYPE may have ATTRIBUTES of 'dbh' and 'height'. An instance of 'tree' also has attributes of 'dbh' and 'height', and not simply a slot for ATTRIBUTES.

4.4.3 Uses of ontologies

This section considers the uses that ontologies may be put to. Most of what follows is derived from an analysis by Uschold (1996). However, an attempt is made to illustrate the material with specific references to ecological modelling and the SYMFOR ontology.

The most important role of the SYMFOR ontology was to provide a rigorous account of the concepts used for building SYMFOR models. The two accounts produced could in theory act as reference material for other modellers. While both the informal (described in Section 4.3.2) and formal (described in Appendix 1) ontologies may be useful in this regard, the informal version is more suitable because it is more readily understood by workers in the domain of ecological modelling. Users of the formal SYMFOR ontology need an understanding of the frame language concepts, KIF and first order logic. This is also the experience of the builders of the Enterprise Ontology (Uschold and Gruninger, 1996).

Ontologies may act as a focus for consensus building within a particular community. If an ontology is to be useful then it must embody concepts that reflect the ideas of the widest number of people possible. This implies that the community of users must agree on the MEaning of individual terms, and the development of an ontology may prove instrumental in obtaining this agreement. A formal definition of each of the terms helps avoid ambiguities. Tools such as Ontolingua, which can be accessed via the World Wide Web (Farquhar *et al.*, 1995) may facilitate this. As one person developed the SYMFOR ontology, it does not provide an adequate test of this.

Ontologies also ostensibly support translation activities. As mentioned in the Section 4.1, one way in which ontologies may be helpful is in supporting the development and use of an interlingua. However, while it is undoubtedly true that an ontology may provide the semantic foundations for translation activities, it must be supplemented with a process of deriving rules or algorithms for translation on the basis of the ontological definitions. Sometimes translation rules may be trivial. This is the case when concepts used in two modelling systems are identical in all but name so that translation may take the form of substituting vocabulary. For example, the concept called a 'level' in Powersim is called a 'compartment' in Stella. Another form of translation may involve 'casting- type' operations in which data are converted to a different form. For example, the precision with which a real number is expressed could be increased or decreased. Other translation rules may be more complex. For example SYMFOR parameters could become intermediate variables with no influences in other systems. In this case there is not a one-to-one correspondence between concepts in the different ontologies so that the derivation of a translation rule is non-trivial.

Development of an ontology may be also have systems engineering applications. The ontology may be used to in the initial specification stage of system design. The development of an ontology may be especially important when a large design team is used, and there is a need to obtain and disseminate a consensus on how the system will operate. Development of an ontology may quickly bring to the surface any ambiguities arising from different designer's perception of the design problem in hand. It may also speed up or 'bootstrap' development of features such as model specification languages. Once the system has been designed and implemented, the ontology may also aid in assessing the design quality. All system components can be checked to make sure that they manipulate data in a manner consistent with the concepts contained in the ontology. For example, a SYMFOR component which allowed or expected state variables to be influenced by intermediate variables would be deemed to be acting in a manner inconsistent with its design. Use of an ontology may also aid the integration of software designed at different times or within different domains. This is because it may be possible to use the ontology to make explicit statements about the assumptions made by the different components. For example, a modelling component may use object-based modules, instead of the algorithm-based modules in use by SYMFOR. This would rule out its use with SYMFOR.

5. Design of SYMFOR software

Software design methodologies make use of different design *viewpoints* to specify the nature of the software being designed. A viewpoint is an abstraction which captures a particular aspect of software design (Budgen, 1995). The two viewpoints most useful with respect to SYMFOR are the *functional* viewpoint and the *dynamic* viewpoint. The functional viewpoint captures the topology of storage, flow and transformation of data within a software system. The dynamic viewpoint captures *behaviour* and *control* in the software system, *i.e.* the way in which the internal state or states of the system change through time in response to specific events, including those initiated by the software user.

In this chapter the functional and dynamic aspects of SYMFOR are described. The descriptions indicate how SYMFOR meets a number of requirements including:

- support for collaborative modelling (Requirement 9);
- support for creation of new Model Designs (Requirement 10);
- support for sharing and reuse of model content (Requirement 11);
- use of a common interface for different models (Requirement 17);
- provision of methods for display of results that can be used with models of different content (Requirement 20);
- provision of meaningful, ambiguous and complete information on models in the system (Requirement 21).

5.1 SOFTWARE DESIGN - FUNCTIONAL VIEWPOINT

The Data-Flow diagram (DFD) for SYMFOR is shown in Figure 5.1. A DFD consists of a series of arrows which join different nodes. Each arc represents a flow of data. Each node in a DFD is either a process, a datastore or a terminator. In this context a process is equivalent to a data transformation, a datastore is a data-set that persists within the system and a terminator is something outside the software system which acts as a sink or source for data.

The DFD illustrates two important SYMFOR features:

- use of formal Model Designs. These offer several advantages (discussed in Section 2.3.1). In SYMFOR these are processed by computer in order to perform three activities: generation of source-code; production of Model Descriptions and creation of datasets for input that are compatible with the model. These activities can be completed much more efficiently than would be the case if they were performed manually.

- use of modularity within the system to facilitate sharing and reuse of model content. In the SYMFOR system there are two module repositories, the 'Module Database' (which is a 'module design repository' in the classification developed in Section 2.3.3) and 'Module Source-code' datastore (a 'module implementation repository'). The modules are stored in source-code and not binary form. This means that they must be incorporated with other Model Source-code then compiled and linked (*i.e.* statically rather than dynamically linked). This was done because of the flexibility it gives in terms of abstract data types. For example, it means that 'trees' can be represented with different data yet still use the same modules (see section 2.3.3).

Figure 5.2 shows how processes are controlled within SYMFOR. It can be seen that two classes of user are responsible for controlling SYMFOR processes: Model Designers and Model Users. Each of these control different processes. The Model Designer controls the Code Generator, the Parameter Generator the Description Generator and the Source-code Compiler. The Model User controls all the displays, the SID compiler and the Model Controller. In some cases the functionality associated with control may be small *i.e.* the user may simply start the process *e.g.* Code Generator. In other cases control functionality may be more sophisticated. For example, Model Users are given a high degree of control over the functioning of SYMFOR displays.

Each of the SYMFOR processes and datastores can be described using Jackson Structure Diagrams (JSDs). The JSD representational formalism is described in Budgen (1994) and Jackson (1983). These diagrams are useful in that they can be used to represent both the structure of process algorithms and the organisation of data in datastores in a diagram. In JSDs, units represented by boxes correspond to components in a sequence. The arrangement and notation of these units capture four details of algorithm/data organisation:

- *decomposition* - this is indicated by positioning one or more sub-units on a level beneath a unit, each one on a stem that originates with the higher unit. Decomposition is used when an activity or data unit can be split up into sub-activities or smaller data units.

- *sequence* - this is indicated by the order in which units on the same level occur, units on the left hand side occur before units on the right hand side.

- *selection* - this is captured by placing a special symbol ('o') inside units on the same level which are mutually exclusive alternatives.

- *iteration* - this is captured by placing a special symbol ('*') inside units which are repeated.

The rest of this section describes each of the terminators, processes and datastores in the system in turn. A brief description is given of each component and JSDs are used as appropriate. Processes in which control functionality is important are briefly described in this chapter with more substantial discussion reserved for the next chapter.

Figure 5.1: Data Flow Diagram for SYMFOR. Ovals represent processes, datastores are represented by the parallel line constructions and rectangles represent terminators. Solid arcs represent flows of data. In the diagram SID is 'Stand Initialisation Dataset', AI is 'Aggregate Information, II is 'Individual Information' and 'FD' is 'Frequency Distribution'.

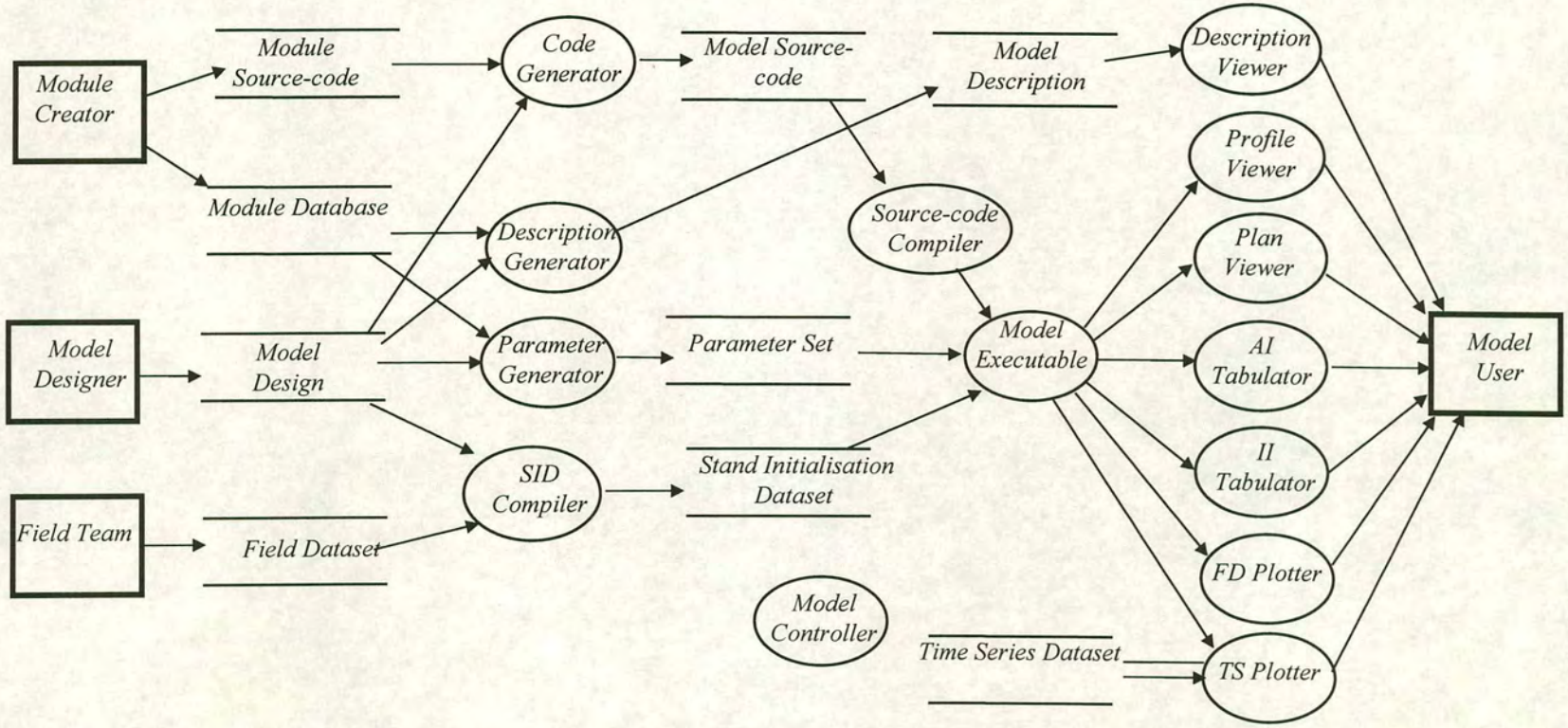
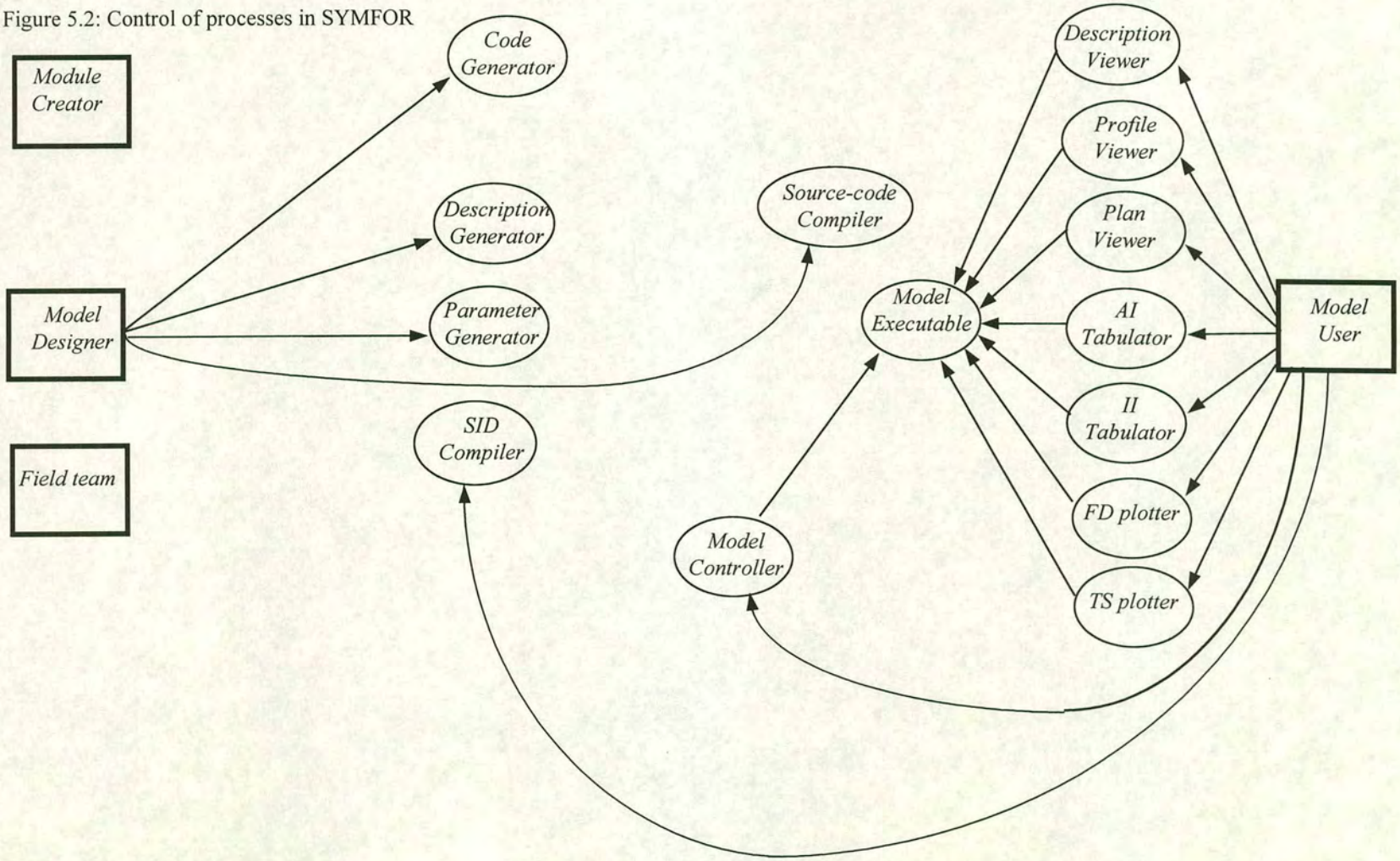


Figure 5.2: Control of processes in SYMFOR



5.1.1 Terminators

5.1.1.1 Model Designer

This terminator is the class of SYMFOR user responsible for designing models. Model Designers use a text-editor to create Model Designs. This means that they must know the principles of individual-based, tree-position simulation modelling. They must also know how to capture Model Design decisions using the SYMFOR Model Design language.

5.1.1.2 Module Creator

This terminator is the class of SYMFOR user responsible for creating new modules. Module Creators use a text-editor to create a Module Source-code datastore. To do this they must have knowledge of programming in C and must understand and be able to follow the programming conventions for creating SYMFOR modules.

Module Creators are also responsible for updating the Module Database by creating a new record for any modules that they have made. This is a text editing operation. To do this they must understand the way in which module information is structured in the database.

5.1.1.3 Field Team

This terminator is the class of SYMFOR user that collects and enters Field Data. They must have knowledge of data collection protocols and be able to enter the data to a computer system in an appropriate format.

5.1.2 Datastores

5.1.2.1 Model Design

This datastore holds details of the content of an individual SYMFOR model. The design is a concise and formal way of storing sufficient information to unambiguously and completely describe a model. There is one Model Design datastore for each SYMFOR model.

The structure of the datastore is shown in Figure 5.3. It contains a series of statements, each beginning on a new line and terminated by a full stop. The statements follow the syntax of the Prolog computer language as described in *e.g.* Malpas (1987) and Sterling and Shapiro (1985). All the statements in the Model Design are Prolog facts: Prolog rules do not occur. Facts consist of a *predicate* followed by one or more *arguments.* The argument list is enclosed in parentheses, and commas separate individual arguments. Representation using Prolog is useful for a number of reasons. First, as a language based on first-order predicate logic, it is recognised as being a very powerful 'knowledge representation formalism'. This means that it is very flexible and can capture many different kinds of knowledge (Muetzelfeldt *et al.*, 1989). Second, it is a pre-existing standard with well-defined syntax rules. Adoption of Prolog syntax therefore avoids any requirement to develop new syntax rules.

Each fact specifies a feature of the Model Design. Often the statement partially or fully specifies an instance of a class defined in the SYMFOR ontology. In this case the predicate will usually correspond to the name of the class in the ontology. For example, an instance of an intermediate variable may be partially specified using:

```
intermediate(tree,height,`m`,`The total height of the tree from base to top of the crown`).
```

In this example the predicate intermediate specifies an instance of the intermediate class defined in the SYMFOR ontology. The modelled entity with which the variable is associated, the name, units and a natural language description of the instance are all arguments in the statement. Other information, such as the position of calculation of the intermediate variable in the order of processing or the algorithm used to evaluate the variable is specified elsewhere in the design. Figure 5.4 shows an excerpt from a Model Design datastore.

A single set of predicates is common to all Model Designs *i.e.* a single set of predicates is *primitive* to all designs. In total there are 15 primitive predicates and they are described in Table 5.1.

SYMFOR Model Designs differ from other text-based model designs in that they are expressed using syntax from a language based on first order predicate logic (*i.e.* Prolog). This means that the Model Designs are directly compatible with Prolog interpreters. This is useful because Prolog supports a number of specialised activities that are difficult to emulate using procedural languages. For example, model interrogation, model checking and even support for model building are all straightforward using Prolog (Muetzelfeldt *et al.*, 1989). With other systems which do not use Prolog compatible representations (*e.g.* those described in Maxwell and Constanza (1997) and Niven (1990)) such functionality would be more difficult to achieve.

Figure 5.3: Jackson Structure Diagram showing the format of a Model Design. It can be seen that Model Designs are made up of a series of facts. Each fact consists of a predicate and one or more arguments. The arguments are either names of model components, real or integer numbers or text strings.

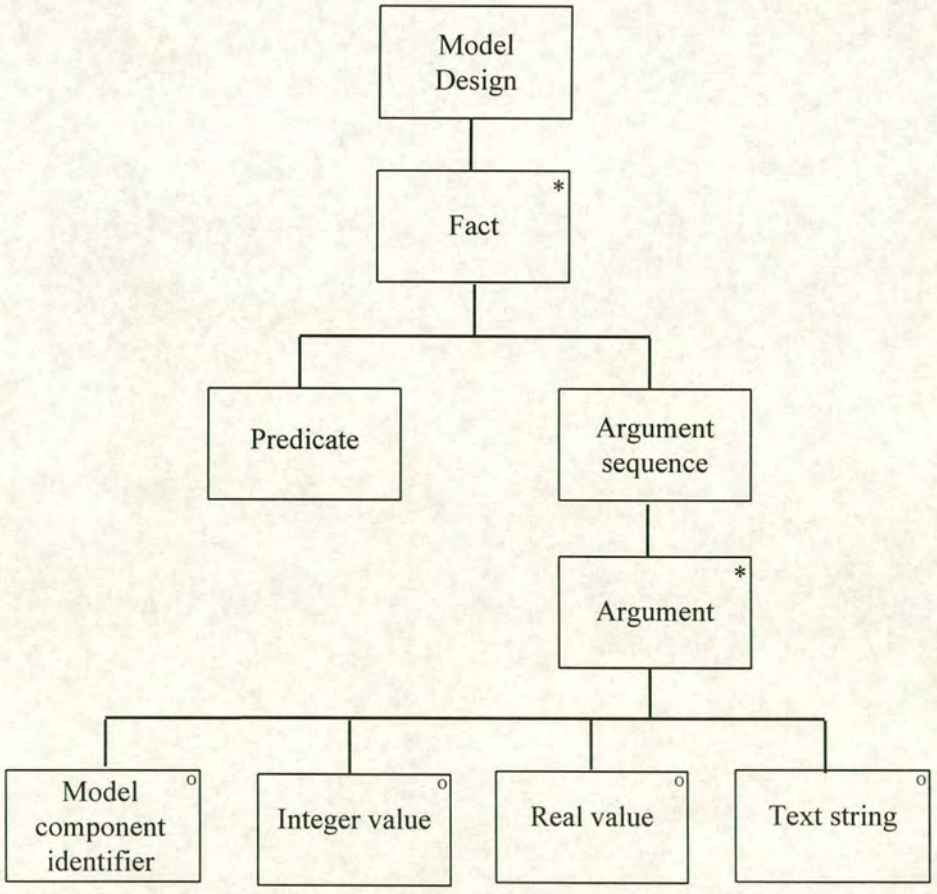


Figure 5.4: Excerpt from a Model Design. The facts follow Prolog syntax with prefix notation *i.e.* each fact is made up of a predicate followed by a list of arguments separated by commas and enclosed in brackets. Facts are each terminated with a period.

```
entity(tree).
pattern(tree,irregular).
stateclassifier(tree, species).
intclassifier(tree,szclass).

statevar(tree,dbh,9.0,1000.0,`cm`,`Diameter of the tree at breast height` ).
statevar(tree,x,-100.0,100.0,`m`,`The x-coordinate of the base of the tree`).
statevar(tree,y,-100.0,100.0,`m`,`The y-coordinate of the base of the tree`).
statevar(tree,year-counter,0,100,`years`,`The number of years the tree has existed in the simulation` ).
intermediate(tree,height,`m`,`The total height of the tree from base to top of the crown`).
intermediate(tree,dbhincr,`cm yr-1`,`The diameter increment of the tree` ).
intermediate(tree,shadeindex,`NA`,`An index of how much competition the tree is facing for light`).
intermediate(tree,treevolume,`m3`,`The merchantable volume of the tree`).
```

Table 5.1: Model Design primitives.

CHARACTERISTIC	DETAILS
Predicate:	creation_date
Syntax:	creation_date(<date>).
	Where <date> is the date on which the design was completed.
Role:	Used to specify the date on which a Model Design was completed.
Predicate:	creator
Syntax:	creator(<author_name>).
	Where <author_name> is the name of the person responsible for designing the model.
Role:	Used to specify the author of a Model Design.
Predicate:	grouping_scheme
Syntax:	grouping_scheme(<grouping_scheme_name>, <ngroups>, <group_list>).
	Where <grouping_scheme_name> is the name of a grouping scheme used in the model, <ngroups> gives the number of classes in the classification, and <group_list> is a list of names for the classes in the classification.
Role:	Used to specify details of classifications used in the Model Design.
Predicate:	entity(<ME_type_name>).
Syntax:	Where <ME_type_name> is the name of a type of modelled entity used in the model.
Role:	Specifies the name of grouping schemes used in a model.

Predicate:	pattern
Syntax:	pattern(<ME_type_name>, <pattern_type>). Where <ME type_name> is the name of the type of modelled entity for which pattern is specified and <pattern_type> is the pattern. This is either regular (number of individual modelled entities is fixed through time) or irregular (number of individual modelled entities varies through time).
Role:	Specifies a characteristic of a type of modelled entity.
Predicate:	statevar
Syntax:	statevar(<ME_type_name>, <sv_name>, <min_val>, <max_val>, <units>, <description>). Where <ME-type_name> is the name of the type of modelled entity, <sv_name> is the name of the state-variable, <min_val> is the minimum value that the state-variable can have, <units> are the SI units associated with the variable and <description> is a text description.
Role:	Specifies the characteristics of a state-variable in the model.
Predicate:	intermediate
Syntax:	intermediate(<ME_type_name>, <intmd_name>, <units>, <description>). Where <ME_type_name> is the name of the type of modelled entity with which the intermdiate variable is associated, <intmd_name> is the name of the intermediate variable, <units> are the SI units associated with the variable and <description> is a text description
Role:	Specifies the characteristics of an intermediate variable in the model.
Predicate:	event
Syntax:	event(<ME-type_name>, <ev_name>, <description>). Where <ME-type_name> is the name of the type of modelled entity with which the event is associated, <ev_name> is the name of the event and <description> is a text description of the event.
Role:	Specifies the characteristics of an event in the model.
Predicate:	conclassifier
Syntax:	conclassifier(<ME_type_name>, <cc_name>). Where <ME_type_name> is the name of the type of modelled entity with which the constant-classifier is associated and <cc_name> is the name of the constant classifier.
Role:	Specifies the characteristics of a constant classifier in the model.
Predicate:	dynclassifier
Syntax:	dynclassifier(<ME_type_name>, <dc_name>). Where <ME_type_name> is the name of the type of modelled entity with which the dynamic classifier is associated and <dc_name> is the name of the dynamic -classifier.
Role:	Specifies the characteristics of a dynamic classifier in the model.

Predicate: Syntax: Role:	delta delta(<ME_type_name>, <sv_name>, <rate_of_change_list>). Where <ME-type_name> is the name the type of modelled entity with which the delta (net rate of change) is associated, <sv_name> is the name of the state-variable with which the delta is associated and <rate_of_change_ list> is a list of the variables constituting the delta. Specifies how state variables should be updated in a model.
Predicate: Syntax: * Role:	modulechoice modulechoice(<entity_name>, <slot_name>, <mc_name>). Where <entity_name> is the name of the type of modelled entity on which the module slot appears, <slot_name> is the name of the capture detail with which the module is associated and <mc_name> is the name of the module choice to be used in the specified slot. Specifies the characteristics of a module choice in the model.
Predicate: Syntax: Role:	cycle cycle(<cycle_name>, <op_list>). Where <cycle_name> is the name of a cycle that occurs in the model and <op_list> is a list of all simulation operations that are performed as part of the cycle. Specifies a cycle used in the model.
Predicate: Syntax: Role:	in_procedure in_procedure(<proc_name>, <entity_name>, <capture_detail_name>). Where <proc_name> is the name of a procedure, <entity_name> is the entity with which an capture detail is associated and <capture detail_name> is the name of a capture detail. Specifies a model procedure.

5.1.2.2 Module Database

This datastore holds details of all the modules present in the SYMFOR module library. Every installation of SYMFOR has only one Module Database. It is similar to the Model Design datastore in that it consists of series of statements each of which follow Prolog syntax. All the statements are Prolog facts, other Prolog constructs such as rules do not occur. There are 5 module data-base primitives, and their details are given in Table 5.2.

The Module Database specifies four pieces of information about each SYMFOR module: the requirements for the module, the name of the file that contains the module implementation, the parameters and a natural language description. The requirements specify details of the Model Design that must be present for the module to be compatible with the design. For example, a module for calculating tree height on the basis of diameter requires tree diameter to be present. Figure 5.5 gives the structure of the Module Database and Figure 5.6 contains an excerpt from a datastore.

In SYMFOR modules are algorithms and do not specify data used in the representation of modelled entities. The reason for this is that it is consistent with the way in which typical users articulate requests for changes in model content. Most frequently the demands for change are in terms of algorithms *e.g.* a user may want to substitute one volume equation for another or to change the logging algorithm. Requests for changes in the set of data used in the representation of a modelled entity are much less common *e.g.* changing the representation of a tree to capture more details of the tree crown. This contrasts with the situation in many other systems where both representation data and algorithms can be specified in modules (*e.g.* Lorek et al, 1998, Smith, 1998; Maxwell and Costanza, 1997)

Modularity satisfies Requirement 11 for SYMFOR, that the system should support sharing and reuse of model content. It also helps to meet Requirement 9, that collaborative modelling should be supported. This is because the efficiency which this allows in exchange of model fragments aids collaborative modelling (Section 2.3.2).

The Module Database is effectively a module design repository in the terminology of the scheme introduced in Section 2.3.3. It stores enough information to let users make an informed choice of the modules that reside in the module implementation repository. SYMFOR differs from other systems in the formality of the descriptions used. In other systems module designs may be informal and produced in the same manner as other software documentation. For example, Lorek *et al.* (1998) provide a diagram to show a hierarchy of objects (modules) available for use in individual models.

Figure 5.5: Jackson Structure Diagram showing the format of the module data base. It can be seen that the data base are made up of one or more module records. The expansion of 'Fact' is the same as the expansion of 'Fact' in Figure 5.3.

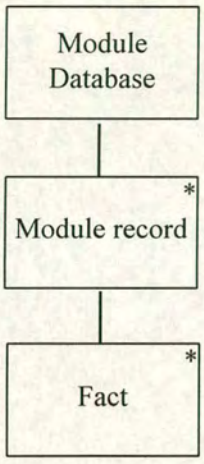


Figure 5.6: Excerpt from a Module Database datastore. The facts follow Prolog syntax *i.e.* each fact is made up of a predicate followed by a list of arguments separated by commas and enclosed in brackets. Facts are each terminated with a period.

```
filename(crownpoint1,`crownp1.sfm`).
evaluates(crownpoint1, tree, crownpoint, intermediate).
parameter(crownpoint1,a,0.55,0.0,1.0,`NA`,`the crownpoint height as a fraction of total height`,[`species`]).
requirement(crownpoint1, tree, height).
description(crownpoint1, `This module finds the crownpoint by assuming that the value is a fixed proportion
        of the total height. The equation used is: crownpoint = height * a where height is the total height
        of the tree and a is a parameter.`).
```

Table5.2: Primitives used in module data-base.

CHARACTERISTIC	SYNTAX
Predicate:	filename
Syntax:	filename(<module_name>, <ds_name>).
	Where <module_name>, is the name of a module and <ds_name> is the name of a datastore containing source-code for the module.
Role:	Used to specify the name of the file which contains the code for implementing a module.
Predicate:	evaluates
Syntax:	evaluates(<module_name>, <entity_name>, <attribute_name>, <att-type>).
	Where <module_name> is the name of a module, <entity_name> is the name of an entity type possessing a slot where the module can go, <capture detail_name> is the name of the capture detail with which the slot is associated and <slot_type> specifies the class of attribute.
Role:	Used to specify the modelled entity attribute evaluated by a module.
Predicate:	parameter
Syntax:	parameter(<module_name>, <par_name>, <default_val>, <min_val>, <max_val>, <units>, <description>).
	Where <module_name> is the name of the module with which the parameter is associated, <par_name> is the name of the parameter, <default_val> is the default value that the parameter takes, <min_val> is the minimum value that the parameter can have, <max_val> is the maximum value that the parameter can have and <description> is a text description of the parameter.
Role:	Used to specify characteristics of a parameter used by a module.
Predicate:	requirement
Syntax:	requirement(<module_name>, <entity_name>, <capture detail_name>).
	Where <module_name> is the name of a module, <entity_name> is the name of the modelled entity with which the capture detail is associated and <capture detail_name> is the name of a capture detail.
Role:	Used to specify Model Design elements that must be present before a module can be used in a Model Design.
Predicate:	description
Syntax:	description(<module_name>, <description>).
	Where <module_name> is the name of a module and <description> is a text description of the module giving full details of how it works.
Role:	Used to specify a natural language description of a module.

5.1.2.3 Module Source-code

The Module Source-code datastore contains the source-code necessary to implement a SYMFOR module. The contents of the datastore are incorporated into model implementation code by a 'cut and paste' type of operation *i.e.* incorporated directly without alteration.

The language used to implement modules and models in SYMFOR is ANSI C (as described in *e.g.* Kelley and Pohl, 1990). C has a number of advantages as a programming language. One of the most important is that it is compiled rather than interpreted so that it runs more efficiently. A second advantage is that it is very widely used. Of particular significance is the fact that large parts of the MS Windows system are written in C. This means that there are fewer technical problems in ensuring compatibility of programs that are written in C with the operating system. It also means that much of the documentation on using the MS Windows Application Programmers Interface uses examples written in C.

C is also a very powerful language. By this is meant that it is possible to undertake many low-level operations that are not possible with high-level languages. For example, it is possible to allocate and manipulate memory directly by use of memory pointers. While this is now regarded as unhelpful for good programming practice, these kinds of capabilities are useful in certain circumstances. Data-structures created in one high level language may be incompatible with data structures created in other languages. When using C, it is easy to put data into a form that a high-level language such as MS Visual Basic can utilise, no matter what form it is used in internally.

The code in each datastore has a well-defined structure, with three different parts (Figures 5.7 and 5.8). The declarations section contains statements which declare (that is allocate storage space for) variables used to store parameter values. Because these declarations take place outside any C functions they have global scope. This means that the storage space persists while the Model Executable process persists. The effect of this is that each time the main module function is called it can use previously stored parameter values. The alternative approach, of reading the parameter values each time the main module function is called, is less efficient computationally. The parameter initialisation function is called when a simulation run starts or when the user changes parameter values. The main function is called for every evaluation that uses the module.

The storage of module implementations in source-code rather than binary form has a number of advantages. First, it means that the requirements that need to be considered when determining whether a module is compatible with a Model Design are simply inputs and outputs. This contrasts with the situation when binary code is used when there may be other considerations. For example, in some systems abstract data types (such as 'tree') must be predefined before a module is incorporated. In other systems, a module may have to possess a particular predefined interface before it can be accommodated (Smith, 1998). The second advantage is that users can inspect the source-code. This aids detection of errors such as inconsistencies between the module design and the module implementation.

Figure 5.7: Jackson Structure diagram showing the structure of a module file. Each module file has three parts: a declarations section, a parameter instantiation function and a main module function.

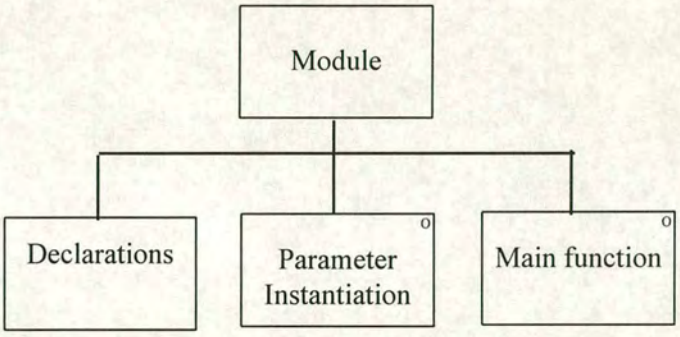


Figure 5.8: Example of a SYMFOR Module Source-code datastore.

```
// Module Dbhincd3.sfm

//----- Describes the relationship between diameter increment and shading index

// DECLARATIONS SECTION
float shdesnsity[NSPECIES][NSZCLASS];
float maxgrowth[NSPECIES][NSZCLASS];

// PARAMETER INSTANTIATION FUNCTION
int dbhincrinit()
{
        if (parinit( "dbhincr3", "shdesnsity",(float __far*)( &shdesnsity[0])) == FALSE) return(FALSE);
        if (parinit( "dbhincr3","maxgrowth", (float __far*)(&maxgrowth[0])) == FALSE) return(FALSE);
        return(TRUE);
}

// MAIN MODULE FUNCTION
int dbhincr( TREE ALLOCTYPE* lptree)
{
        int szclass;
        int sps;
        float expo;

        szclass = lptree->szclass - 1;
        sps = lptree->species - 1;
        if (lptree->shadeindex == 0.0f) lptree->shadeindex = 0.001f;

        expo = -shdesnsity[sps][szclass]* (float)( log((double) lptree->shadeindex)) +
        maxgrowth[sps][szclass];
        lptree->dbhincr = (float)(pow(10, expo)) /10.0f;
        if (lptree->dbhincr < 0.0f) lptree->dbhincr = 0.0f;
        return(TRUE);

}
```

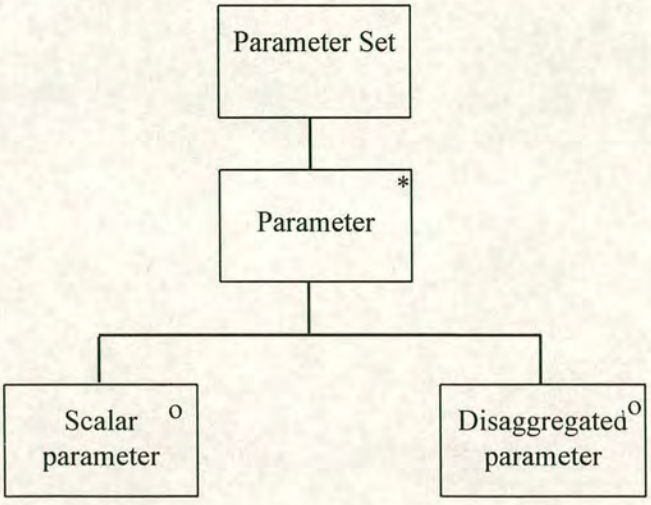
5.1.2.4 Parameter Set

The Parameter Set datastore contains the complete set of parameter values required to run the model with which the datastore is associated. There are two kinds of parameter in SYMFOR (Section 4.2) and each is represented in the Parameter Set datastore in a different way (Figures 5.9 and 5.10). Scalar parameters are parameters which have only one value. In this case the parameter is represented by a single line in the file consisting of an identifier followed by the value. The identifier is made up of the module name with which the parameter is associated conjoined with the name of the parameter. Disaggregated parameters have several associated values. They are represented by several lines in the parameter datastore, one line per value.

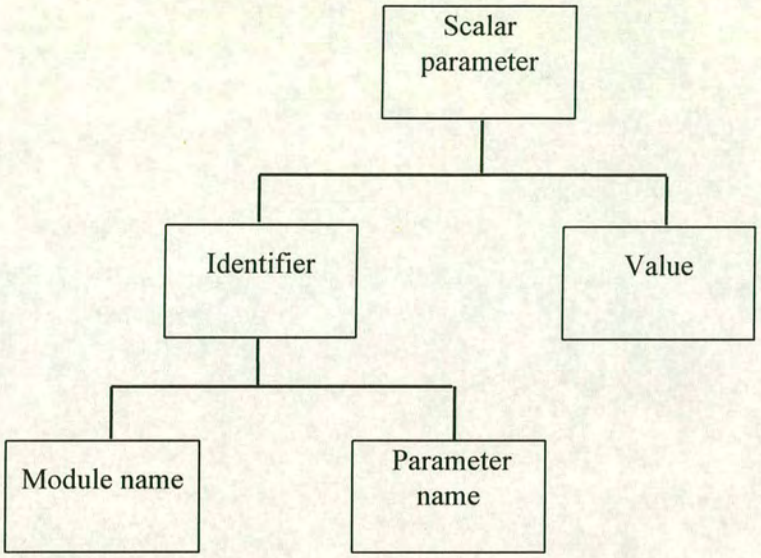
In SYMFOR the number of parameters in a Parameter Set may vary according to the model in use. In addition there is no requirement that parameters should come in a particular order. Other forest simulation models such as GHAFOSIM (described in Alder, 1995) also use model input files to facilitate the entry of data used in model calculations. However, in a GHAFOSIM parameter file a particular sequence of values must be strictly adhered to, and the same number of values must be present in each file. In addition values may not be tagged, so that it may be very difficult to interpret the content of a parameter file without reference to accompanying documentation or the source-code of the model.

In modelling environments such as Stella, POWERSIM and AME information on model content, initial stand structure and parameter values may be stored in a single file. Moreover, the file may be binary and so not editable outside of the modelling environment. The latter feature was requested by stakeholders.

Figure 5.9: Structure diagram showing the format of a Parameter Set datastore.
a) Top level



b) Expansion of scalar parameter



c) Expansion of disaggregated parameter

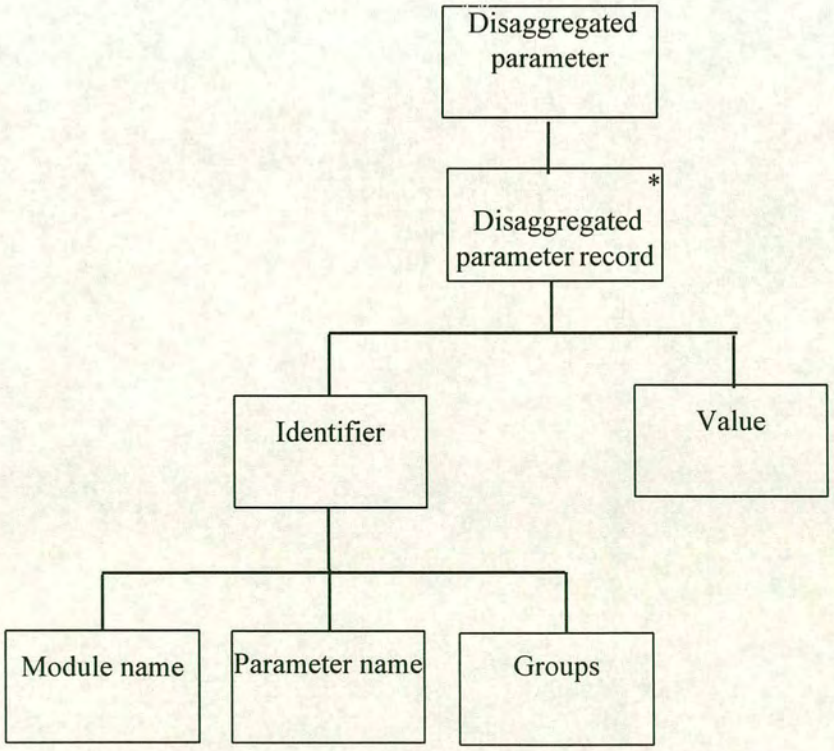


Figure 5.10: Excerpt from a parameter file.

```
treevolume2:formfactor= .45
foliage1:folstartslope= 800
foliage1:folmax= 400
damageintensity2:npoints= 10
stageinc1:stageincLAI0[species( 1)]= 2.5
stageinc1:stageincLAI0[species( 2)]= 2.5
stageinc1:stageincLAI0[species( 3)]= 2.5
stageinc1:stageincLAI0[species( 4)]= 2.0
stageinc1:stageincLAI0[species( 5)]= 2.5
stageinc1:stageincLAI0[species( 6)]= 0
```

5.1.2.5 Model Description

This datastore contains a natural language description of the Model Design. Each statement in the datastore is an English sentence. The datastore is created by the Code Generator process using information from the Model Design and the Module Database. Figure 5.11 shows an excerpt from a Model Description datastore.

The description follows a precise structure. General information on the model (such as the creation date and the author) is followed by a description of the way in which different kinds of modelled entity are represented. The data used in the representation and the algorithms used in the model for calculating values for represented data are given. Units of all values are indicated.

The Model Description complements the formal Model Design. While the formal design is unambiguous, complete and concise, its terse syntax lowers its readability to people unfamiliar with the formalism. The Model Description is also unambiguous and complete but is much more comprehensible (as it is in the form of natural language). In addition, the Model Description includes details of modules used in the model (Model Designs indicate the modules that have been selected, but do not have any detail of the contents of these modules). For these reasons the Model Description component of SYMFOR satisfies 21 (*The framework should provide information on each model in the system that is meaningful, unambiguous and complete).*

One problem with the Model Description is its length. While the design is complete, it is also very long. This makes it difficult to navigate through so that finding a piece of information of interest can be time-consuming. This could be addressed by using a hypertext Model Description. This would, for example, list the modelled entities present and have hyper-links to details of each individual modelled entity.

5.1.2.6 Stand Initialisation Dataset (SID)

This datastore contains values that are used to initialise the stand at the start of a model simulation (*i.e.* to specify the modelled entities present and the values of their state variables and modelled entity constants). Each datastore contains information on one or more types of modelled entity (*e.g.* tree, gridsquare *etc*). For each type there is a value indicating the number of individuals of that type present and a dataset containing data pertaining to individuals of the type. Each individual has a sequence of values associated with state variables, a sequence of values associated with modelled entity constants and a sequence of values associated with constant classifiers. The structure of the datastore is shown in Figure 5.12.

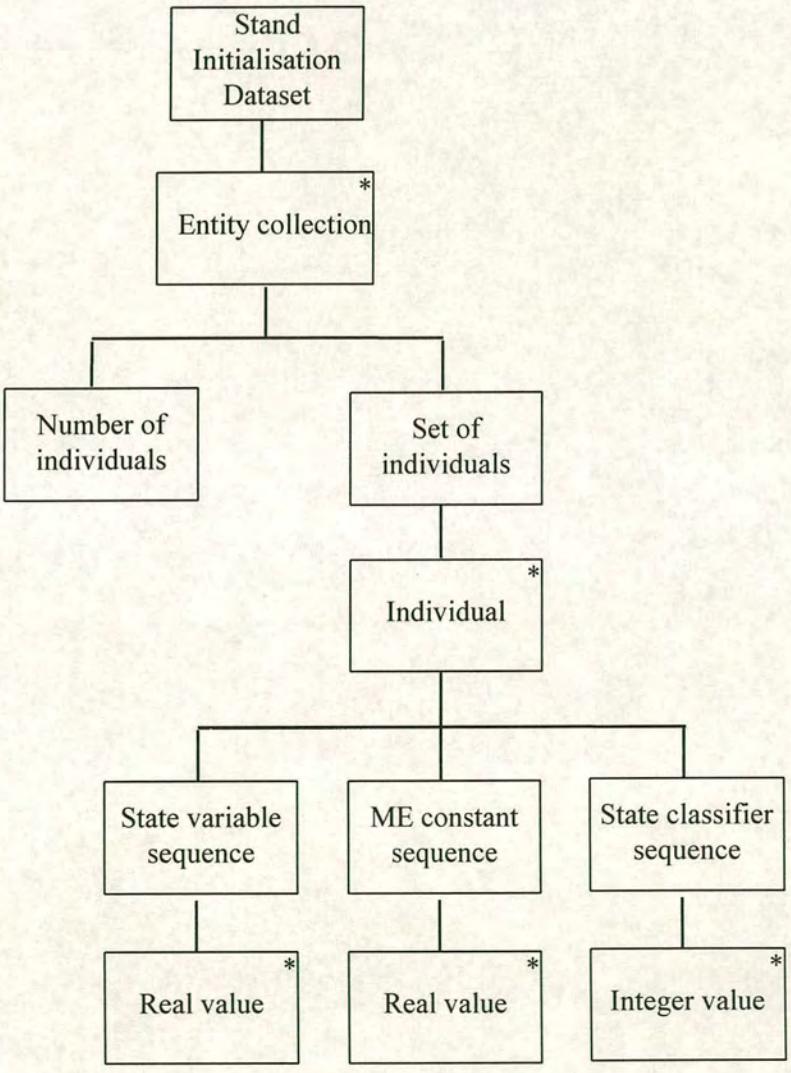
The SID is stored in binary form. This form of data can be read into the Model Executable much more efficiently than the equivalent text form. Although the fact that it is binary means that it cannot be accessed *e.g.* with a spreadsheet or text-editor application, in practice this is not a problem because the data all originate in Field Dataset datastores, and these are directly readable.

Figure 5.11: Excerpt from a Model Description datastore

```
1.   GENERAL INFORMATION

This model was created by allen on 20/5/97.
At any point in a simulation modelled entities of 7 different types
may be present.
These are trees, fallentrees, stands, gridsquares, felledtrees,
skidtrails and seedsap_cohorts.


2.  ENTITY INFORMATION


2.1   TREE INFORMATION
Trees vary in number through time.

Trees have 4 state variables.  They are:

   dbh - Diameter of the tree at breast height.  (cm)
   x - The x-coordinate of the base of the tree.  (m)
   y - The y-coordinate of the base of the tree.  (m)
   age - The age of the tree.  (years)

Trees have 9 intermediate variables.  They are:

   height - The total height of the tree from base to top of the
       crown.  (m)
       height2 is the module used to calculate height. This module
       finds the height of a tree based on its diameter.  The
       relationship used has the form of a rectangular hyperbola.
       This module has 2 parameters. They are:
           maxheight - the height at which response to diameter
               saturates.  Note that this value is usually about
               10 m more than the maximum height observed in the
               field. (m)
           This parameter is disaggregated by species.
           startslope - the initial slope of the response
               function. (m)
           This parameter is disaggregated by species.

   dbhincr - The diameter increment of the tree.  (cm yr-1)
       dbhincr3 is the module used to calculate dbhincr.  This
       module finds the value of diameter increment based on the
       shadeindex of a tree.  As shadeindex increases dbhincr
       decreases.  The relationship is linear.
       This module has 2 parameters. They are:
           shdesnsity - the sensitivity to shading - the slope
               of the line describing the relationship
               multiplied by -1. (cm yr-1)
           This parameter is disaggregated by species and by
           szclass.
           maxgrowth - the growth when shadeindex = 0. (cm
               yr-1)
           This parameter is disaggregated by species and by
           szclass.

   shadeindex - An index of how much competition the tree is
       facing for light.  (NA)
```

SIDs persist after they are created. This means that the same SID can be used repeatedly without having to be recreated each time. Repeated use of the same SID is required for run replication, an essential part of any simulation experiment (when models contain stochastic elements). Creation of SIDs is time-consuming and involved, (Section 5.1.2.7) so that it may be difficult to consistently produce the same file without error.

Figure 5.12: Jackson Structure Diagram showing the format of a Stand Initialisation Dataset (SID).

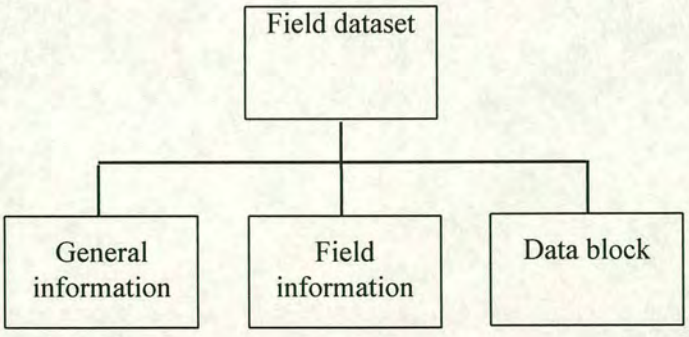


5.1.2.7 Field Dataset

This datastore holds values derived from field observations on a set of individual entities. The information in one data set relates to a single forest stand. All the entities in the datastore are of the same type of modelled entity. For example, data from observations on trees are stored in a single data set, and cannot be mixed with data from gridsquares. The format is Comma-Separated values (csv), *i.e.* ASCII with the convention that data in columns are separated by commas and data rows are terminated with carriage returns. Figure 5.13 shows the structure of a Field Dataset used to store data from observations made on individual trees in a stand, while Figure 5.17 shows an excerpt from a Field Dataset.

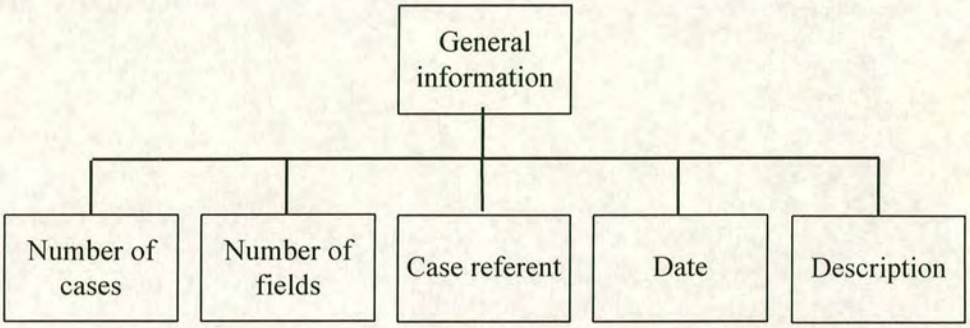
Each datastore has three parts: general information, field information and a data block. The general information section gives details that relate to the entire set of measurements in the dataset, for example the stand from which the measurements were taken and the date(s) on which the observations were made. The field information section gives information on the different kinds of observation that were made on individuals. For example it may specify that 'stem diameter' was one of the observations made on trees in the stand. The data block section contains the actual values of the various fields for individuals in the stand. The top-level structure of the datastore is captured in Figure 5.13.

Figure 5.13: Jackson Structure Diagram showing the format of a Field Dataset datastore



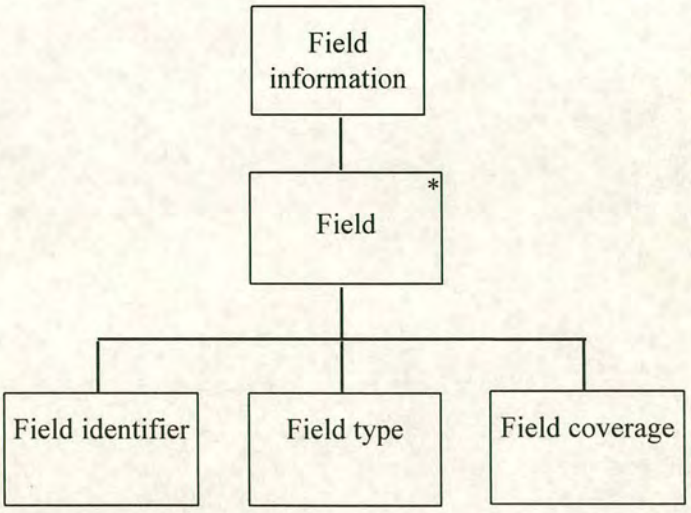
The detailed structure of the general information part of the datastore is shown in Figure 5.14. It can be seen that this section contains five items of information, each of which is scalar. The number of cases gives the number of individual entities described in the datastore. The number of fields, gives the number of observations that could potentially have been made on each individual represented in the datastore. The case referent, is the type of the individuals described in the datastore. The date gives the date(s) on which observations were made while the description gives details of the stand in which the observations were made (for example, latitude and longitude or plot identifier).

Figure 5.14: Jackson Structure Diagram showing the format of the General information section of a Field Dataset datastore.



The field information part of the datastore specifies the nature of the data produced by various observations of the individual entities. For each observation the coverage (whether there is a value for every individual in the datastore) and the type of datum resulting from the observation is given (Figure 5.15). Allowable types are real number, integer and text.

Figure 5.15: Jackson Structure Diagram showing decomposition of the field information part of the Field Dataset datastore



The final part of a Field Dataset is the data-block. The structure of this part is shown in Figure 5.16. It holds the data associated with observations of individual entities. Each row in the block holds information relating to a single individual. Columns correspond to the fields described in the field section while rows correspond to individual entities. One or more blank values may be present in fields associated with an individual.

Figure 5.16: Jackson Structure Diagram showing the decomposition of the data-block part of the Field Data datastore

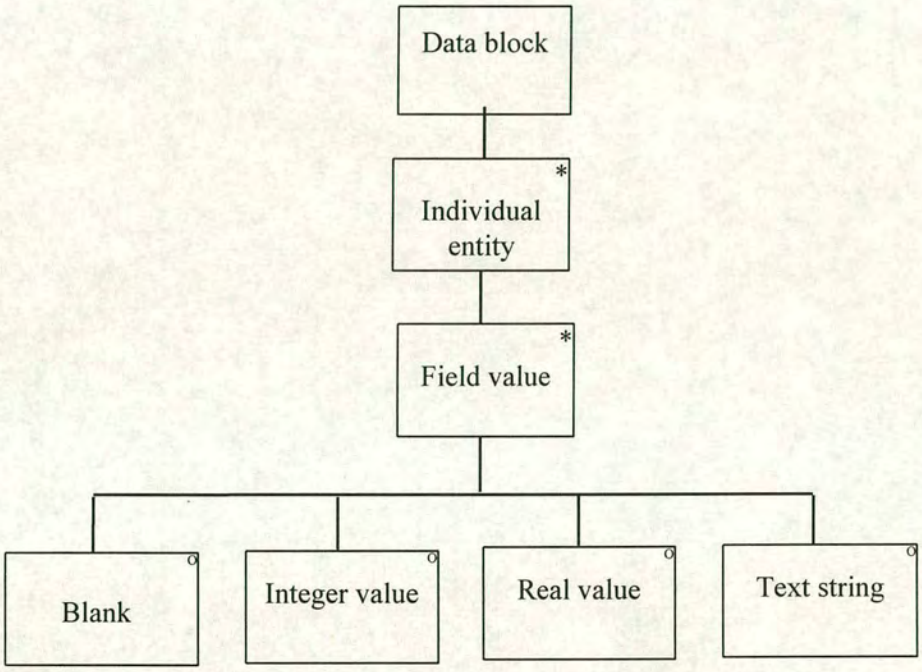


Figure 5.17: Excerpt from a Field Dataset. Note that the Field Dataset is shown loaded into a spreadsheet for clarity. If it were in a text editor then columns would be separated by commas.

General information

Field information

Data block

	A	B	C	D	E	F	G
1	ncases	669					
2	ncolumns	18					
3	icase	tree>10 cm dbh					
4	date	Oct-93					
5	description	Data comes from 1 ha plot (Plot5) in Wanariset Sangai Research Forest					
6	T.N	real	complete				
7	GRT	real	complete				
8	DIAM	real	complete				
9	POM	real	complete				
10	CQ	integer	complete				
11	USG	real	partial				
12	CP	real	partial				
13	MHT	real	partial				
14	USD	real	partial				
15	THT	real	partial				
16	X	real	complete				
17	Y	real	complete				
18	DAYAKNM	string	complete				
19	FAMILY	string	complete				
20	GENUS	string	complete				
21	SPECIES	string	complete				
22	AGE	real	complete				
23	SPSGRP	integer	complete				
24	1	49.5	15.8	130	3		10.8
25	2	291.5	92.8	320	4	202	29.6
26	3	125.5	39.9	220	3	101	20
27	4	50.4	16	130	3		9.3
28	5	83.7	26.6	170	3	70.5	21
29							

The format of the Field Dataset is such that it can accommodate many different kinds of dataset. The number of fields and the number of observations can vary from datstore to datastore. Datastore can also be used to store partial sets of observations, *i.e.* sets which do not possess a record for all the individuals for which other observations are made.

The handling of large volumes of data in text format is not ideal because of the large size of the data files and slow access times. A better solution may be to make use of a commercial Relational Database Management System (RDBMS) such as MS Access, MS FoxPro or Borland DBASE. These have a number of advantages: data is stored in binary format - files are much smaller and data can be accessed much more quickly; it is easier to manage inter-related datasets that are related to each other in some way; special functionality is available for inspecting and managing data. Alder (1995) discusses the use of RDBMs in the context of managing forest inventory data in more detail.

5.1.2.8 Time Series Dataset (TSD)

This datastore contains data generated by individual model simulations. It consists of a rectangular data set where rows correspond to simulation time points and columns correspond to run output such as total volume or volume in a particular size class. The format of the datastore is comma-separated value (csv). A single datastore can hold the results from many simulation runs. Successive runs are stored sequentially in the datastore. This allows comparison of replicate runs (which is useful when models have stochastic elements) and comparison of different treatments. Figure 5.18 shows a typical TSD.

The Model User rather than the Model Designer specifies the set of output that is entered in the datastore by the Model Executable. The Model User specifies output to be stored from the using the functionality of the Model Controller. This process is then responsible for obtaining raw output from the Model Executable, deriving the output required and writing them to the TSD.

As TSDs are in a text-based format they can easily be read-into specialised applications such as spreadsheets or packages for statistical analysis. This means that specialised tasks can be performed without having to explicitly program these tasks in SYMFOR.

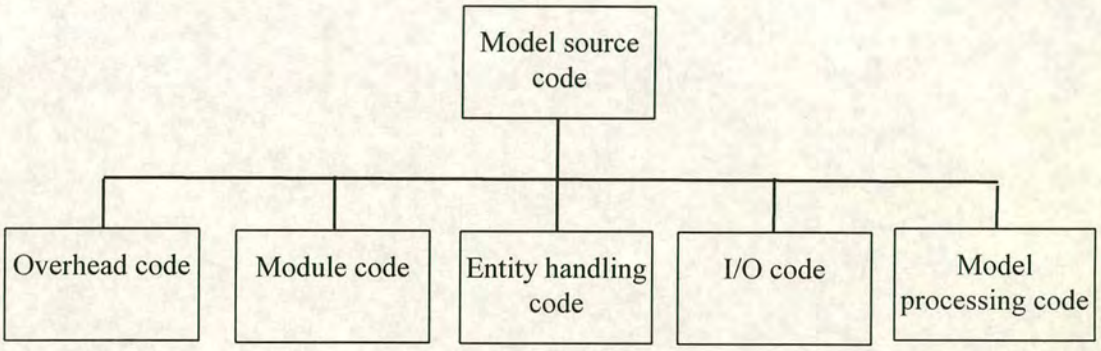
Figure 5.18: Excerpt from Time Series Dataset (TSD)

Run	Time	Standing trees	Total volume	Total basal area	Timber production	yearly harvest
1	0	479	298.42	31.97	0	0
1	1	491	298.75	32.03	0	0
1	2	497	299.26	32.03	0	0
1	3	503	299.50	32.03	0	0
1	4	529	300.61	32.30	0	0
1	5	532	299.66	32.16	0	0
1	6	538	298.20	31.99	0	0
1	7	574	299.72	32.41	0	0
1	8	593	300.51	32.48	0	0
1	9	609	301.75	32.63	0	0
1	10	638	303.40	32.97	0	0
1	11	714	306.00	33.75	0	0
1	12	707	308.28	33.69	0	0
1	13	700	309.50	33.77	0	0
1	14	697	297.50	32.79	0	0
1	15	698	299.09	32.96	0	0
1	16	707	301.24	33.21	0	0
1	17	704	302.77	33.37	0	0
1	18	697	303.94	33.44	0	0
1	19	697	304.87	33.50	0	0
1	20	696	306.63	33.69	0	0
1	21	695	308.28	33.86	0	0

5.1.2.9 Model Source-code

This datastore holds the source-code that is compiled to create a Model Executable. It consists of a series of statements in the C computer language. There is one Model Source-code datastore for every model in the SYMFOR installation. Figure 5.19 shows the top-level structure of the Model Source-code datastore.

Figure 5.19: Jackson Structure Diagram showing the structure of a Model Source-code datastore



Overhead code is code that is not directly related to the Model Design but that is needed for the target executable to function within its host operating environment. Programs running under Windows 3.x must all designate at least one window to receive messages from the operating system. This is achieved by having two functions: a window initialisation function, called on program entry and an associated *call-back* function to process messages sent to the main window. In SYMFOR these two functions have little functionality.

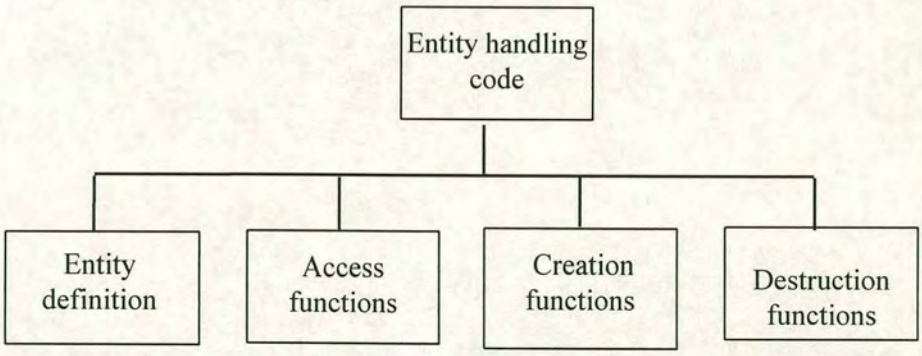
Module code is code associated with the Model Designer's choice of modules. Modules are used to specify the calculations that should take place when intermediate variables and events are evaluated. The structure of modules in code is exactly the same as the structure of the module in the Module Source code datastore (described in Section 5.1.2.3).

Entity handling code is code that is used to manage modelled entities and their data in a simulation. The structure of this part is shown in Figure 5.20. Entity data is stored and manipulated using the construct of a 'structure' that is available in C. There is a different entity definition for every type of modelled entity present in a model. Modelled entities of the same type are held in *collections,* and *handles* are used to identify and manipulate individual entities within each collection. Handles are integers which uniquely identify individual entities in a collection.

Access functions are used to retrieve a memory pointer to a modelled entity data based on its handle. The memory pointer can then be used to directly read or modify modelled entity data. This is a common strategy in operating systems (such as MS Windows) where memory is dynamically allocated and moved around so that memory pointers may become out-dated. There is one access function for every modelled entity collection in the model.

Destruction functions are used to destroy individual entities, while creation functions are used to create new individual entities in a simulation. Creation functions reuse handles that have been liberated by modelled entity destruction. They also may dynamically allocate more memory as and when appropriate for storage of modelled entity data.

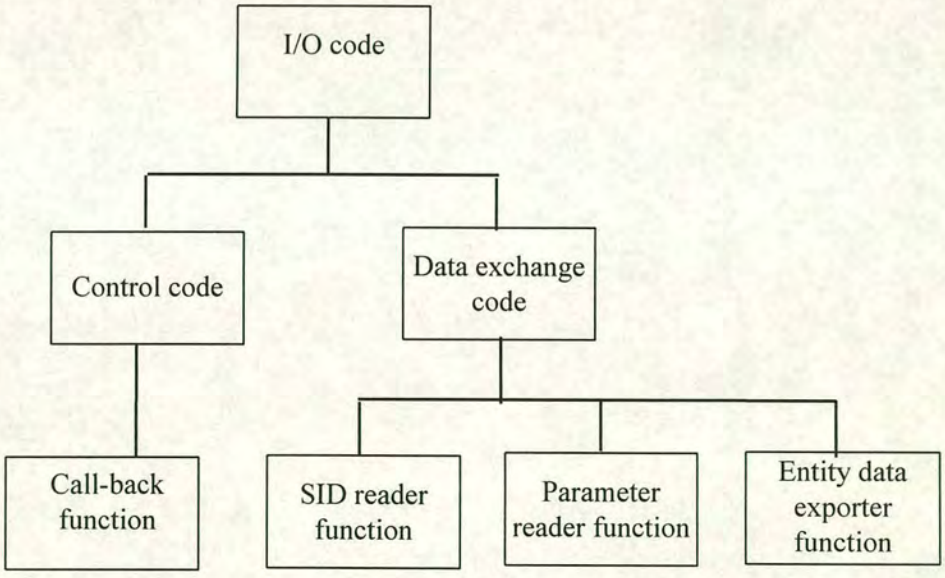
Figure 5.20: Jackson structure diagram showing the structure of the modelled entity handling code found in Model Source-code



I/O code is responsible for the interaction between the Model Executable and its environment. Two types of interaction can occur: model control and data exchange (Figure 5.21). Model control is an input that is received by the Model Executable and that originates with the controller of the simulation. The Model Executable performs operations in response to model control input. These operations include inputting or outputting of data, updates or model calculations.

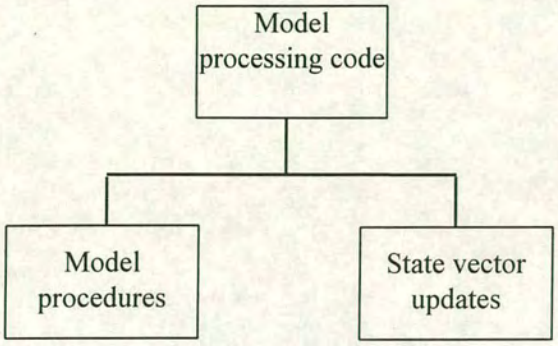
Data exchange code implements input and output of data in response to model control input. Data input comes from two datastores: the Parameter Set from which parameter values are obtained and the SID from which initial values for modelled entity state-variables and ME constants are obtained. Values for modelled entity attributes are exported by the Model Executable to display processes.

Figure 5.21: Jackson Structure Diagram showing the decomposition of the 'Generate I/O code' block.



Model processing code determines how the model responds to control input. There are two classes of processing operations: model procedures and updates (Figure 5.22). SYMFOR procedures are described in detail in Section 4.2. To summarise, they consist of a series of evaluations of attributes that are always performed as a unit. The attributes in a procedure may be intermediate variables, intermediate classifiers or events. Updates are model operations in which the model state-vector is revised to take account of the model calculations that have occurred since the last update.

Figure 5.22: Jackson structure diagram showing decomposition of the model processing code.

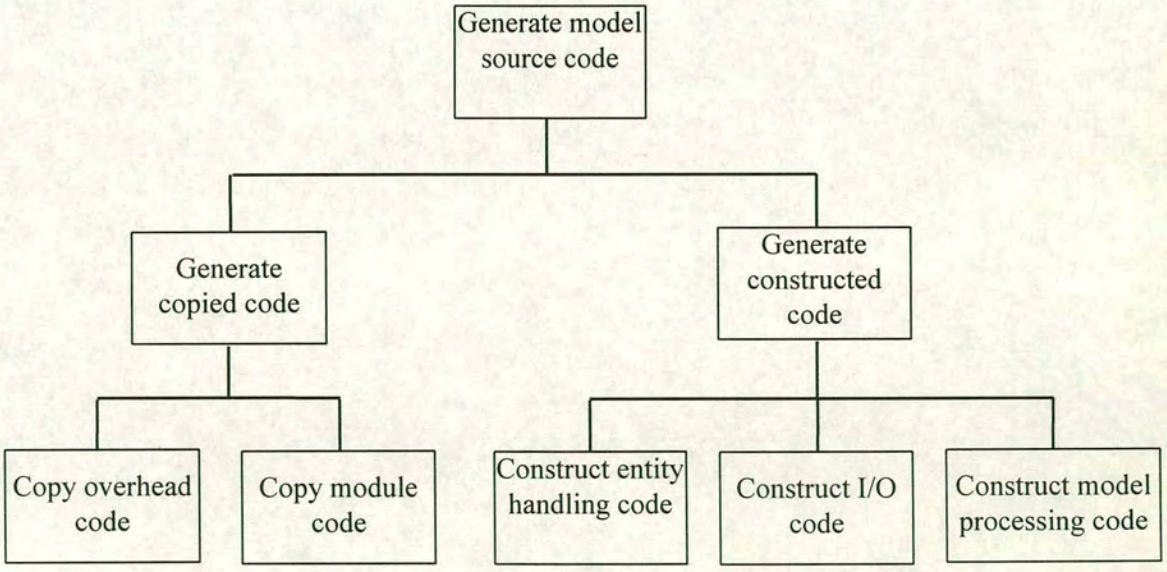


5.1.3 Processes

5.1.3.1 Code Generator

This process is used to produce Model Source-code datastores. It uses information from the Model Design datastore and from one or more Module Source-code datastores to do this. Figure 5.23 shows the structure of the algorithm used by the Code Generator process.

Figure 5.23: Jackson Structure Diagram showing the structure of the Code Generator process



The Code Generator uses the two methods of *copying* and *constructing* in code generation. Copying involves a 'cut-and-paste' type operation in which module source-code is inserted directly into Model Source-code. Constructing involves employing special algorithms to create code on a line-by-line basis. Construction of code is necessary to provide sufficient flexibility to deal with Model Designs of different content within the framework. For example, every Model Executable contains functionality for reading in model parameters, either at the start of a simulation or in response to the user changing one or more parameter values. However, the set of parameters used by a model will depend on the modules in use, and this may vary from model to model. For this reason a different parameter reader function must be used in each Model Executable. The structure of the appropriate function is determined by the parameters used by the model, which is in turn determined by the modules in use. The Code Generator consults the Model Design and Module Database to obtain this information before generating source-code for the appropriate function.

Both module code and overhead code are copied. This is possible because the statements within these code blocks do not depend upon the content of the Model Design. Overhead code is the same for every model and module code is the same in every model in which the module choice with which the code is associated is present.

A different strategy based on specification and use of *idioms* in source-code must be followed for constructed code. These are conventions for interpreting and processing information in Model Designs so that appropriate functionality in source-code is created. For example, an idiom for creating SYMFOR procedures may specify that:

- each procedure specified in a Model Design is implemented as a function in the Model Source-code. Procedures specify a set of compound evaluations that take place as a unit. For example, a procedure called 'growth' may contain compound evaluations for 'tree height', 'tree shadeindex' and 'tree diameter increment'.

- each function contains source-code for performing all the compound evaluations associated with the procedure. A compound evaluation involves evaluations of an attribute for all individuals of a type. For example, a compound evaluation of tree height would involve the evaluation of tree height for every individual tree represented in a model.

Idioms may be further refined to specify the nature of individual lines of source-code. For example every function must start with a particular sequence of text (which gives the function name) and end with a particular sequence of text. Idioms are important because they form the basis for Code Generator algorithms.

The relationship between idioms and Code Generator algorithms is best illustrated by using an example. Figure 5.24 shows a source-code function suitable for implementing the 'growth' procedure described above. There are several features of this function that are important with respect to creating a Code Generator algorithm. First, it can be seen that the internal structure of the procedure depends upon details in the design (*i.e.* the compound evaluations within the function depend upon the definition for the procedure found in the Model Design). This means that some parts of the function show an iterative structure.

Figure 5.24: Structure of function used to implement model procedures

```
void  growth_proc()                                          header

{

  int   status;

                                                          declarations
  int   htree ;
  TREE *lptree;

  for (htree=1; htree<=treecollection.UpperActiveHandle; ++htree  ){
     lptree = Usetree( htree , &status );
     if ( status == ACTIVE ) {                            compound_eval[1]
     if (height( lptree )  == FALSE) return;
        }
     }

  for (htree=1; htree<=treecollection.UpperActiveHandle; ++htree ){
     lptree = Usetree( htree , &status );
     if ( status == ACTIVE ){                             compound_eval[2]
     if (shadeindex( lptree )  == FALSE) return;
        }
     }

  for (htree=1; htree <= treecollection.UpperActiveHandle;++htree  ){
     lptree = Usetree( htree , &status );
     if ( status == ACTIVE ){                             compound_eval[3]
     if (dbhcinr( lptree )  == FALSE) return ;
        }
     }

}                                                            close
```

The Code Generator works by first establishing an output stream between the Code Generator and the datastore into which generated code is sent. Statements are then sent to this stream by the Code Generator as it runs. Pseudocode for a Code Generator algorithm suitable for implementing SYMFOR procedures is shown in Figure 5.25. It can be seen that there are three different kinds of statement:

- print statements. These statements all start with 'print'. They are used to send *string constants* and *string variables* to the output stream. String constants are enclosed in quotes, while string variables end in '$'. String constants are used for source-code text that does not change each time a code-generation algorithm is used. For example, each procedure function always starts with the text 'void' (to indicate that it does not return a value). For this reason 'void' is a string constant in the code generation algorithm. In contrast, the name of the function being created will be different every time the code-generation algorithm is run. For this reason it is handled as a string variable (procnm$).

- design interrogation statements. These statements all start with 'retrieve'. They are used to procure information from the Model Design. They are responsible for assigning values to string variables (which are later sent to the output stream). Interrogation of the Model Design is used to ensure that relationships specified in the design are reflected in source-code. In the example discussed previously, the design specifies that a procedure called 'growth' is responsible for performing three compound evaluations. To correctly produce appropriate source-code the subroutine retrieve_cmpd_evals is called. The first argument procnm$ is instantiated with the name of the procedure currently being processed (in the'growth' example this is assumed to have been done in statements preceding those shown). This argument acts as an input to the sub-routine. The second and third arguments are instantiated within the subroutine (*i.e.* they act as outputs from the subroutine). n_cmp_evals is a variable which is assigned a value corresponding to the number of associated compound evaluations and cmp_evals$() is an array where each element is assigned the name of a compound evaluation that is associated with the procedure.

- looping statements. The statement starting the loop begins with 'for', while that indicating the end of the loop begins with 'next'. Loops are used to create parts of the source-code that are essentially iterative in nature. For example, in the growth example, code to loop over all tree modelled entities is repeated for every compound evaluation. This is achieved by the use of a for-next loop in the code-generation algorithm.

Figure 5.25: Pseudocode for the Code Generator algorithm responsible for creating code to implement SYMFOR procedures. 'retrieve_cmpd_evals' is a function that returns the compound evaluations associated with a procedure (the name of which is given by proc_nm$). 'n_cmpd_evals' is the number of compound evaluations returned and cmpd_eval_nm$() is an array containing the names of returned compound evaluations. 'g_procfn_eval' is a function that takes the name of a single compound evaluation as an input which it generates source-code for.

```
print "void " ;procnm$; "_proc()"
print "{"
print " int status;"
retrieve_proc_metype( procnm$, met$)
print " int h"; met$; ";";
print ucase$(met$); " *lp"; met$; ";"
retrieve_cmpd_evals( procnm$, n_cmpd_evals, cmpd_eval_nm$( ) )
for i = 1 to n_cmpd_evals
    gen_proc_eval(cmpd_evals$( i))
next i
print "}"
```

Different idioms and code-generator algorithms can be used to produce source-code for different design features. For example, different idioms are appropriate for coding updates, parameter reader functions and SID reader functions. However all of them can be captured in code generation algorithms using the elements described above *i.e.* design interrogation, looping and writing to an output stream.

Model realisation through automatic generation of source-code is discussed in Section 2.3.1. In general it is more complex than model realisation through use of design interpreters as used in *e.g.* Stella or AME. However, one of the most important advantages is that the runnable model will execute efficiently. This is because the use of an interpreter to manage simulations inevitably decreases the efficiency of execution. Moreover, as a low-level language (C) is used to specify algorithms, execution should be especially efficient.

Another advantage is that the system is very good at coping with changes to the Model Design over time (see Section 2.3.2). There is a very short length of time between change to the design and production of a runnable model, compared with a situation in which design changes were manually implemented. This flexibility in the system satisfies SYMFOR Requirement 10, which states that the system should be able to support new Model Designs.

5.1.3.2 Description Generator

The Description Generator uses the information contained in the Model Design and information contained in the Module Database to generate a natural language description for a model. The approach used is very similar to that used for the Code Generator process *i.e.* algorithms (based on appropriate description idioms) interrogate the Model Design and Model Database and send a stream of text to a datastore.

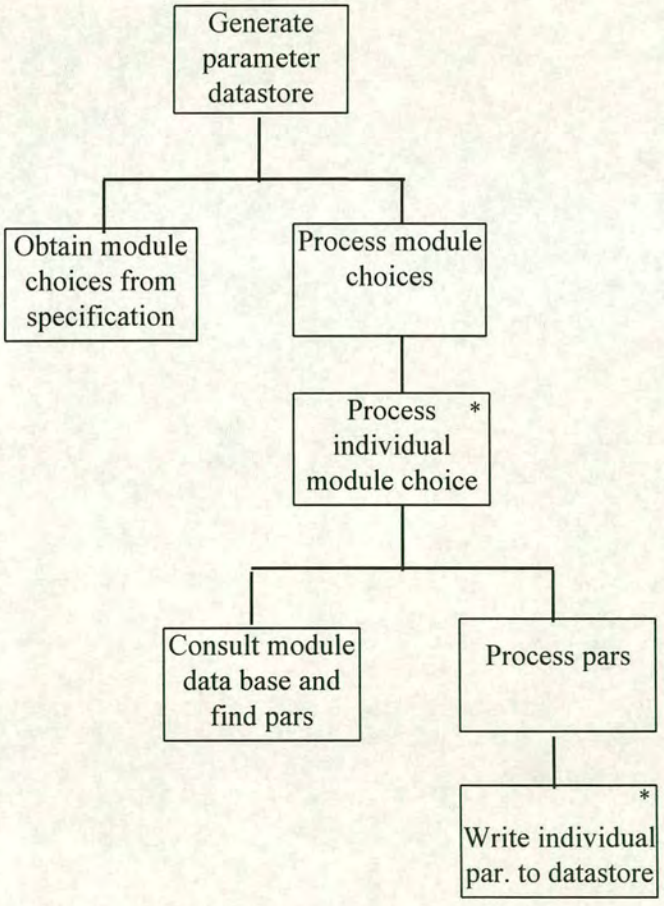
Automatic description generation is useful for the same reason that automatic code generation is useful: it allows rapid production of new descriptions after Model Designs are changed. It is also less likely to lead to error. In a manual system, changes to model documentation and model content may be made at different times, or even by different people. This can lead to a high chance of inconsistencies arising between the two (Lorek and Sonnenschein, 1998).

5.1.3.3 Parameter Generator

The Parameter Generator uses the choice of modules specified in the Model Design and the information in the Module Database to generate a set of parameters and parameter values for a model. This is required in a system such as SYMFOR in which different models are used, as not all models will share the same set of parameters.

The algorithm used is represented in Figure 5.26. The procedure decomposes into two sub-procedures. First, the module choices associated with the model are found from the Model Design. Second, the parameters associated with this set of module choices are found. Consulting the Module Database to find the parameters associated with each module choice in turn does this. Parameters that are found are then written to the parameter datastore.

Figure 5.26: Jackson Structure Diagram showing the structure of the algorithm used by the Code Generator



Automatic generation of appropriate Parameter Sets is useful in that it avoids parameter redundancy. Often flexibility is achieved in stand-alone models by incorporating alternative subroutines, each of which has a different set of parameters. The user is then asked to choose the particular subroutine that they want to use when the model runs. One problem with this approach is that at any one time many of the parameters are redundant, and it can be difficult to work which parameters are actually in use.

5.1.3.4 Source-code Compiler

The Source-code Compiler acts a process in SYMFOR. It produces a Model Executable using Model Source-code as input. The 'compiler' preprocesses, compiles and links the code. A standard commercial compiler (Microsoft Visual C++ compiler version 1.0) is used in SYMFOR.

There are a number of advantages of using a commercial compiler compared to a freeware alternative. Commercial compilers will generally have better user-support. Commercial compilers may be more generally available from vendors. Commercial compilers may be more up to date. Commercial compilers may be more reliable, as the company producing them may be legally responsible for the quality of the product.

The main disadvantage is cost. Two types of cost may be incurred. First, there is the outlay that must be made for the initial purchase of the compiler. This can be quite steep, of the order of £100 or more. Second, there is spending associated with obtaining frequent upgrades of the compiler. Most commercial companies regularly produce new versions of their product. While there may be not be an absolute requirement to purchase a new version when it becomes available, it is desirable for a number of reasons: older versions may cease to receive user support; older versions may become unavailable and older versions may not be able to utilise new features of operating systems to optimal effect.

Disadvantages associated with cost of commercial compilers are less severe because not all SYMFOR users will require to use the compiler. The analysis presented in Section 3.1 indicates that some stakeholders do not actually need the ability to change Model Designs. Both forestry trainers and forest operations managers are in this category. Even when stakeholders require the ability to change Model Designs, it is possible that this could be undertaken by transmission of new designs to a central unit responsible for the activity. For example, forest concession researchers could pass new designs to a Government research department for implementation.

The software architecture used in SYMFOR is such that substitution of one compiler for another should be comparatively easy. This is so because full details of the models are explicitly stated in Model Designs and the concepts used to express details are not related to a particular compiler. This means that the same design could be submitted to a different code-generator to produce code compatible with a different compiler. While a rewrite of the Code Generator may be required this may not be a major task. Difficulties only arise because of extensions to ANSI C that are used to handle *e.g.* memory management in model source code. Extensions to ANSI C will typically differ between different compilers, even if they are used to achieve the same activities.

5.1.3.5 Model Executable

The Model Executable holds modelled entity data and performs the various operations that are used in model simulations. It can be said to be an implementation of the Model Design. The Model Executable acts as both a process and a datastore in SYMFOR. It acts as a datastore in that it is associated with a file that persists in the system (the executable file). When this file is run it performs transformations of input data (*i.e.* model operations) so that in this respect it is like a process.

The Model Executable stores functionality for undertaking model runs in binary form. One ostensible disadvantage with this is that the contents of the runnable model cannot be directly inspected. However, in SYMFOR this is not a problem because two other representations of model content are used: the formal Model Design and the Model Description. In addition, the source-code used in the creation of the Model Executable can be retained for inspection.

The approach used in SYMFOR makes the process of distributing models slightly more complex than in other systems. Even though the Model Executable is a stand-alone program in that it can be launched and run in an operating system independently of other programs it is not 'stand-alone' in other respects. One reason is that it cannot be used without a Model Design. This is because the Model Design is consulted by the Model Controller and the SYMFOR displays to determine the control statements that the Model Executable will respond to. For example, the model controller must know the names of the model operations that the Model Executable can perform before it can send requests to the Model Executable.

There are several advantages stemming from the separation between Model Executables (which house model content) and the model interface (*i.e.* the model controller and the model displays). First, it means that interface components need only be coded once. It also means that the same interface can be used with models of differing content. This is SYMFOR Requirement 17 (*There should be a common interface for all models within the framework.).*

Second, it means that model interface can evolve independently of the underlying models. It also means that different interfaces can be used to run the same model. This is useful because different classes of user may interact with models in different ways. For example, forestry students might be expected to make full use of visualisation tools. Others may make use of an interface with less scope for visualisation but more functionality for conducting simulation experiments *e.g.* functionality for sensitivity analysis and statistical comparison of treatments. Bundling all functionality together in a single interface may prove confusing and may make the software more difficult to use.

5.1.3.6 Model Controller

The model controller is unique amongst the SYMFOR processes in that its primary role is control of another process (the Model Executable) rather than transformation of data. Its main job is to facilitate user control of simulations. It can issue four different kinds of instruction to the Model Executable:

- read SID - this is done at the start of simulations.

- read Parameter Set - this is done at the start of simulations or when the model user changes a parameter value;

- perform procedure - this is done as part of each cycle (*i.e.* iteration) of a simulation. There may be several procedures in a cycle. Procedures are defined in section 4.3.2.

- perform update - this is done as part of each cycle.

Advantages of separating the model controller (which is primarily an interface) from the Model Executable (which is primarily model content) were discussed in the last section. As a large amount of model controller functionality is dedicated to providing a user interface it is more fully discussed in the next Chapter.

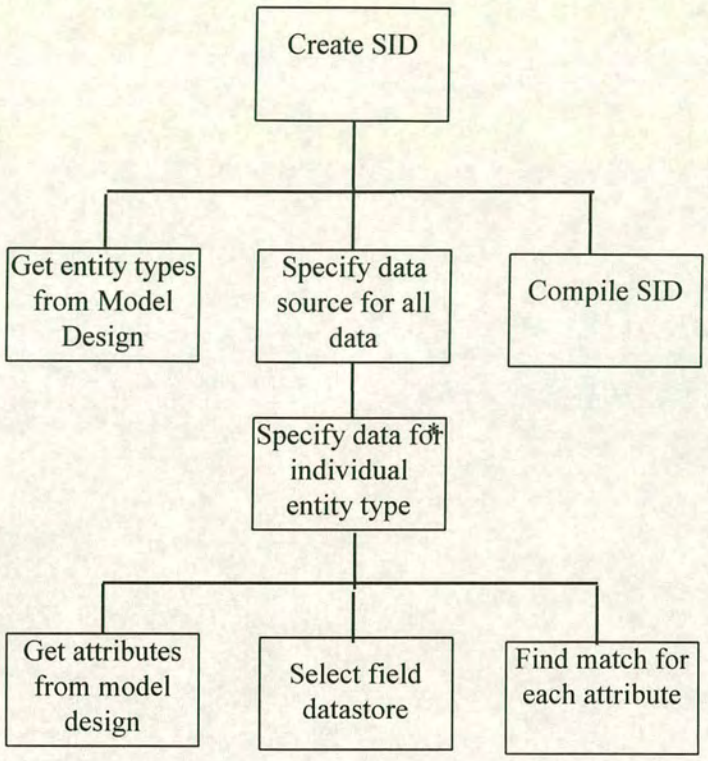
5.1.3.7 SID Compiler

The SID Compiler uses information from one or more SYMFOR data files to build a SID datastore that is suitable for use with a particular model. The procedure involves specifying a dataset for each kind of modelled entity in use by a model, then joining these together. Three operations are involved for each type of modelled entity: the selection of a suitable Field Dataset, matching fields in the datastore with attributes of the modelled entity, and the output of data from Field Data records to the SID. The procedure is summarised in Figure 5.27.

Two advantages of the method used are that names of fields in the datastore need not match names of modelled entity attributes and there is no need for consistency between different Field Datasets. In the process of creating an SID users have to explicitly match fields with modelled entity attributes. While this is time-consuming, it means that there does not have to be a general standard for nomenclature. The production of such a standard is difficult because the data collected from field plots may vary. This is especially likely to be the case in the future as new models with different data requirements may be created.

While some validation of data is currently undertaken, there is scope for improving validation in future. At the moment users can only select fields for which there is full coverage, and they can only match a field to a modelled entity attribute if they agree in type. For example, data on stem diameter cannot be matched with a tree species attribute. In future it might be possible to specify realistic ranges in value for modelled entity attributes, and only allow matches with Field Datasets in which data fall within these ranges.

Figure 5.27: Jackson Structure diagram showing the structure of the algorithm used by the SID c Compiler.



5.1.3.8 Profile Viewer

The Profile Viewer processes information on tree modelled entities supplied by the Model Executable to produce a diagram showing the modelled stand in cross-section. In these diagrams ellipses are used to represent tree crowns and lines are used to represent tree stems. The trees are sorted so that those closest to the viewpoint are superimposed on those further away. Section 6.2.1.15 - 16 give more detail on the conceptualisation and implementation of this display.

5.1.3.9 Plan Viewer

The Plan Viewer processes information on modelled entities to produce a diagram showing locations of various modelled entities in the x, y plane of the modelled stand. Sections 6.2.1.13 - 14 give more details on the conceptualisation and implementation of this display.

5.1.3.10 Aggregate Information Tabulator

The Aggregate Information Tabulator processes information on modelled entities supplied by the Model Executable to produce a table of values. Each value is an aggregate statistic *i.e.* each value is determined by an operation on a set of individual modelled entities. Sections 6.2.1.10 - 11 give more details on the conceptualisation and implementation of this display.

5.1.3.11 Individual Information Tabulator

The Individual Information Tabulator produces a rectangular table of values for a particular modelled entity type. Each row in the table corresponds to an individual of the modelled entity type selected. Attributes pertaining to the individual entities are arranged in columns. Section 6.2.1.12 gives more details on the conceptualisation and implementation of this display.

5.1.3.12 Frequency Distribution Plotter

The Frequency Distribution Plotter processes information on modelled entities supplied by the Model Executable to produce a frequency distribution. The frequency distribution is a histogram where a series of vertical bars are used to represent the number of individual modelled entities in a series of classes. The length of each bar is directly proportional to the number of individuals in the class.

5.1.3.13 Time Series Plotter

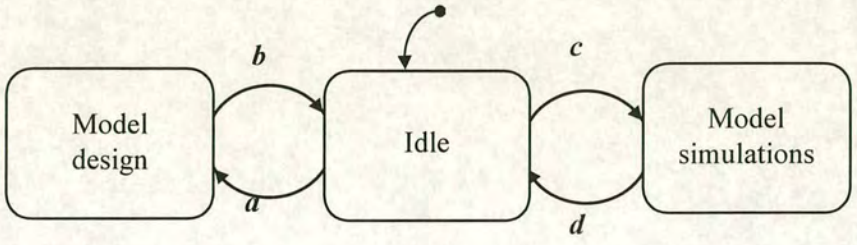
The Time Series Plotter processes information in the TSD to produce graphs showing how TS variables change through time. Time is always on the x-axis. Section 6.2.1.1-20 give more detail on the conceptualisation and implementation of this display.

5.2 SOFTWARE DESIGN - DYNAMIC VIEWPOINT

The dynamic behaviour of SYMFOR is captured in a series of *statecharts* (Budgen, 1994). Figure 5.28 shows that SYMFOR can exist in one of three states: idle, model creation mode and a model simulation mode. When SYMFOR is idle it means that no SYMFOR activity is taking place.

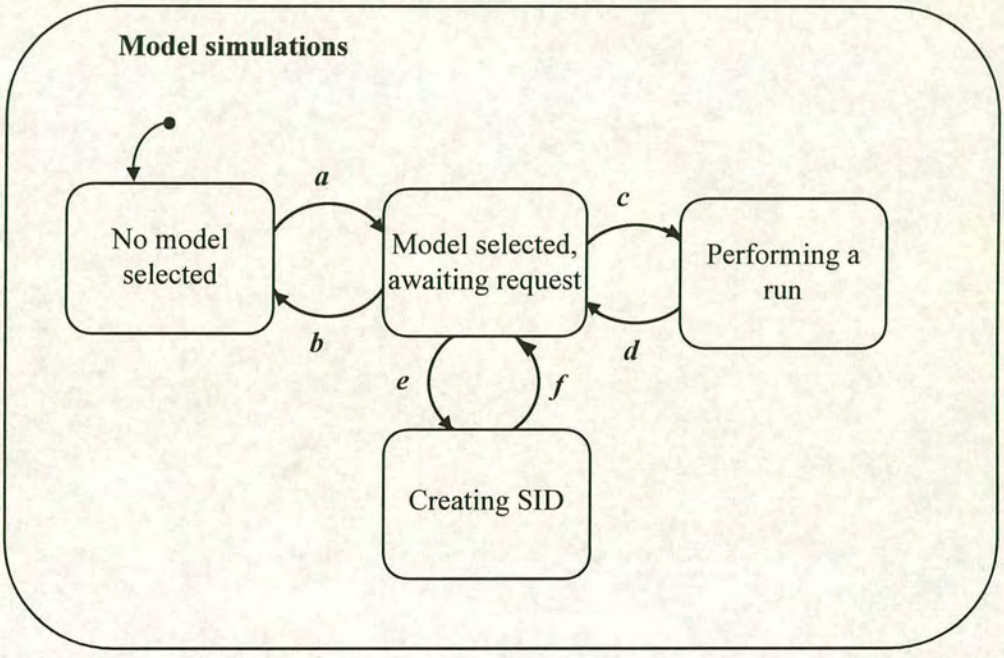
The activities of designing models and running models are not tightly integrated in SYMFOR. The model creation state SYMFOR allows users to create new models, and the model simulation state allows users to carry out model simulations.

Figure 5.28: Statechart describing SYMFOR. *a* occurs when a user starts any design software, *b* occurs when a user closes design software, *c* occurs when a user starts software for running the model, *d* occurs when the user closes software for running models



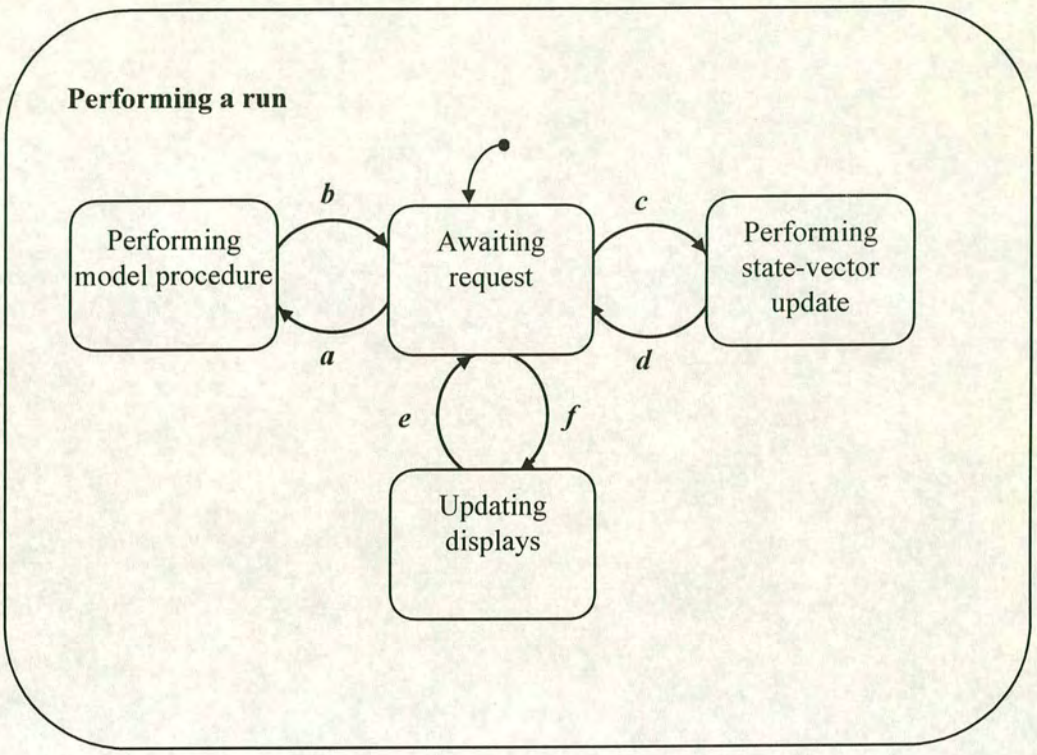
The 'model simulations' state encapsulates 4 states: 'no model selected'; 'model selected, awaiting request' and 'performing a run'. This is the situation depicted in Figure 5.29. The user chooses a model to work with in the transition between 'no model selected' and 'model selected'. The transition between 'model selected' and 'run initialised' requires the user to select two datastores: a SID and a Parameter Set.

Figure 5.29: An expansion of 'Model simulations' state from (a). ***a*** occurs when the user opens a model, ***b*** occurs when the user closes a model, ***c*** occurs when the user starts a run, ***d*** occurs when a user chooses to finish a run, ***e*** occurs when the user chooses to create a new SID and *f* occurs when the user aborts or finishes the creation.



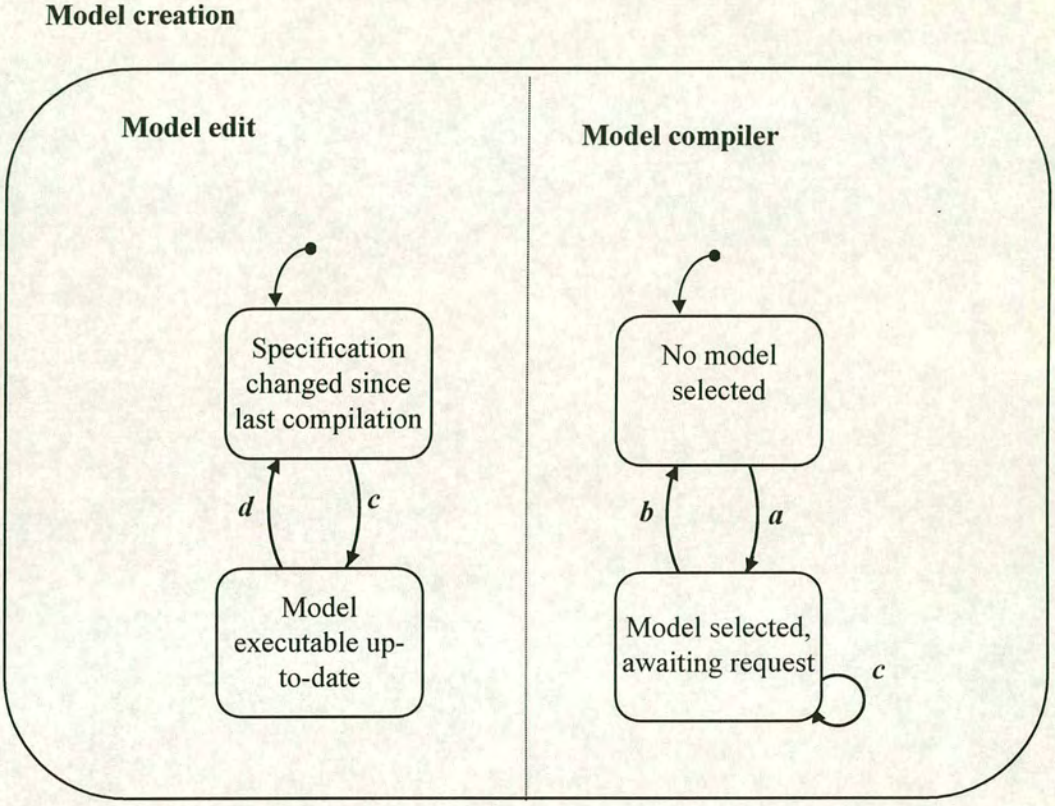
While in the state of 'performing a run' model procedures and updates are used to project the modelled stand into the future. The statechart for this is shown in Figure 5.30. It can be seen that there is an intermediate state, awaiting request between the two states of performing model calculations and performing update. The transition out of this state is controlled by requests from the model controller. The transitions to this state occur automatically when the action being performed has terminated.

Figure 5.30: Elaboration of the 'performing a run' state. *a* occurs when a calculation request is received by the model controller process, *b* occurs automatically when the calculation is finished, *c* occurs when an update request is received from the model controller and *d* occurs automatically when update is finished.



The statechart for model creation is show in Figure 5.31. It can be seen that the state of model creation is a compound of the state 'model edit' and the state of the model compiler. One event is common to both specification and compiler, that of model compilation.

Figure 5.31: Elaboration of the model creation state. Event *a* is triggered by the user opening a model in the compiler. *b* is triggered by the user closing a model. *c* occurs when the user compiles the model and is common to model design and model compiler. *d* occurs when the user edits the Model Design.



5.3 DISCUSSION

SYMFOR was designed using a methodology derived from that of Structured System Analysis and Design (SSAD) rather than more modern 'object-oriented' methodologies. By this is meant is that the primary design effort went into identifying datastores and processes and constructing a DFD. Recently many object-oriented techniques have gained favour, so that it might be thought that this would be most appropriate for the design of SYMFOR rather than the more antiquated SSAD. However it is widely recognised that multiple viewpoints may be appropriate in the design of software. For example, Rumbaugh *et al.* (1991) describe a methodology ('Object-oriented Modeling and Design') in which 3 viewpoints are used in software development: the *object*, *functional* (which corresponds to the viewpoint used in Section 5.1) and *dynamic* viewpoints which corresponds to the viewpoint used in Section 5.2). Rumbaugh *et al.* state that for any one application these viewpoints may not be equally important, the viewpoints of importance being determined by the nature of the application.

The nature of SYMFOR is such that the object viewpoint is essentially trivial. One reason for this is that there are no inheritance relationships between components of the system. For example a Model Design is simply a Model Design, it is not a specialisation of a more generic class. In addition the compositional hierarchy is trivial: a single Model Design has a number of associated components such as a Parameter Set, a Model Description etc.

One feature of the SYMFOR design that is not desirable *per. se.* is its complexity. SYMFOR can be said to be complex in that there are a large number of interacting components in the system - in all there are 27 datastores and processes in the DFD. In more traditional modelling approaches there may be only 3 components: an input file, a model executable and an output file (e.g. the GHAFOSIM system described by Alder (1995). Such systems are initially faster to implement as they avoid any difficulties associated with handling the interaction of many different components.

There are however several very important advantages of using the SYMFOR approach. Most of these relate to the flexibility with respect to model content that the more complex design affords. For example, in SYMFOR data from Permanent Sample Plots (PSPs) is stored in two datastores: the Field Dataset and the SID. This is not desirable in that the same data is held twice, and so requires twice as much space on the storage medium used than if only a single datastore was used in the system. However the Field Dataset holds PSP data in a generic form and is readily accessed and processed by the SID compiler to produce SIDs that are compatible with particular Model Designs. In the traditional modelling approach change in model content may require substantial alteration of input files - for example adding values for newly incorporated state-variables. This is typically undertaken on an *ad hoc* basis with limited computer support for validation and hence greater scope for human error. In general the more complex design of SYMFOR means that changes to model content can be handled much more effectively.

The design described in this chapter can be said to be the *logical* design of SYMFOR. This is because it is at a high level of abstraction. This is useful because it brings to the surface the conceptual basis of the system. There are however many different ways in which this design could be implemented. For example, the logical design does not specify the platform or software development systems to be used. The *physical* design supplements the logical design and specifies details of this kind, and it is this that forms the basis of the next chapter.

6. Implementation of SYMFOR software

This chapter describes the implementation of the SYMFOR software. Implementation specifies how the design described in the previous chapter is realised. In particular it specifies how applications incorporating the different processes of the data flow diagram were created. The programming and interface of each application is described.

Two development tools were used in the implementation of SYMFOR: MS Visual Basic (VB) and MS Visual C++ (VC). Each of these is superior to the other in certain respects. VB is a high level language that features strong support for building of Graphical User Interfaces (GUIs). It is however computationally inefficient (by virtue of being interpreted rather than compiled) and does not enforce highly structured programming. Visual C++ is computationally efficient, supports structured programming and is very powerful. It is however a lower level language so that the instructions required to perform an activity are longer and more esoteric than those required to accomplish the same activity in VB.

VB provides a number of special constructs for GUI development including:

- the *form* - this is a rectangular window on which other GUI elements can be placed in the interface design process;
- *control* - this is a GUI elements that vary in functionality and visual characteristics and that are placed on forms in the interface design process;
- *program module* - this is a group of VB procedures. Procedures in the same module can access and manipulate a common set of data not accessible outside the module.

In MS Windows two things determine the characteristics of a form or control:

- the *class* of the control. In VB a set of different control classes is supplied. Each of these has characteristic functional and visual characteristics. List-boxes and push-buttons are examples of control classes. VB does not support *super-classing* (the creation of new classes based on existing classes).
- modification of class properties and functionality in *instances* of a control. This is referred to as *sub-classing* and is supported by VB.

Although not specifically an Object Oriented language, the VB programming language has some of the functionality often associated with Object Oriented languages including:

- programming objects (forms and modules) in which data and procedures can be logically associated;
- encapsulation of data within programming objects;
- strong support for the event-driven programming metaphor;

Implementation of SYMFOR involved building three programs: two MS Windows *applications* (*i.e.* programs that have windows capable of directly receiving and responding to user actions such as double-clicking) and one Dynamic Link Library (or DLL *i.e.* a library of functions that can be used as a resource by different clients but that cannot directly receive or react to user actions).

The two SYMFOR applications were built using VB. The SYMFOR Model Compiler (SMC) application is used to build individual SYMFOR models while the SYMFOR Model Manager (SMM) application is used to conduct model simulations. Neither of these programs performs intensive computation associated with running models such computation of shading indices and one (the SMM) has a highly developed GUI. For these reasons implementation in C is not particularly appropriate.

The SYMFOR DLL was built using VC. The DLL is used to perform some tasks that cannot be performed in VB (such as retrieval of data from a model executable) and tasks that are commonly used in model executables (such as random number generation). The reasons that VC were used are technical in nature. They relate to the fact that VB does not support dynamic linking of procedures *i.e.* it is not possible for a VB program to export its procedures to more than one client in the same way as can be done with functions in programs written in VC. The reason why this is important is described in Section 6.4.

The material in the sections describing the SYMFOR applications is split into Interface and Programming. The Interface part specifies interface design from the point of view of a user of the software. The Programming section indicates the way that code which was not specifically related to the implementation of the user-interface was structured. In the case of the SMM and SMC applications the Programming section describes the VB program modules that were used.

6.1 SYMFOR MODEL COMPILER (SMC)

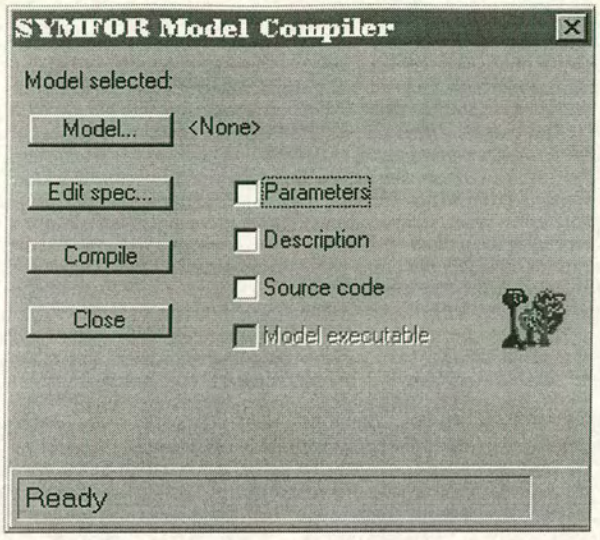
This application is used to perform activities associated with the creation of a new model, rather than activities associated with running a model. It was created using MS Visual Basic Version 3.0. It implements three of the processes from the SYMFOR DFD: Code Generator, Parameter Generator and Description Generator.

6.1.1 Interface

The user-interface for the SMC is very simple. It consists of a single window that appears when the application is loaded. There are four push-buttons on the window. The push-button labelled 'Model' opens a dialog-box for selecting a model. The dialog-box is identical to that used in the SMM and is described in Section 6.2.1.9. After a model has been selected, the name of the model appears on the window (it replaces the '<None>' label). The push-button labelled 'Edit spec' is used to open a text-editor loaded with the formal model design associated with the model. The push-button labelled 'Compile' is used to start the compilation process, while the push-button labelled 'Close' is used to exit the SMC application.

The activities undertaken as part of the compilation process are set using the check-boxes. Any or all of the three activities of parameter generation, description generation or source-code generation can be selected. The progress with compilation is indicated in the status-box at the bottom of the window.

Figure 6.1: The SMC Application



6.1.2 Programming

There are seven program modules in the SMC (Table 6.1). One of these contains code for controlling the SMC, three of these contain code for the performing the compilation activities, and three contain functionality for handling the various datasets used by the SMC. Because of the simple design of the SMC, control functionality is fairly trivial. The three compilation activity modules contain functionality for source-code generation, parameter generation and description generation. The algorithms for undertaking these activities are described in Sections 5.1.3.1 - 5.1.3.3.

Functionality for handling individual data sets used by the SMC is contained in separate VB modules. Most of the datasets used correspond to data stores in the SYMFOR DFD described in Chapter 5. For example, there are modules for handling model design data and module design data. Each program module stores data associated with the data set and is interrogated or modified by other parts of the SMC that need to access the data. One advantage of this approach of maintaining an *internal representation* of data is that data files need to be consulted less frequently (for example, at the start and end of a session). As accessing an internal representation (which is usually stored in the computer's main memory) is much faster than file access this means that the SMC is more efficient than an alternative which does not use internal representations. Another advantage is that data is 'encapsulated' (*i.e.* subject to restricted access) within the module so it is much more difficult to unintentionally alter data in other parts of the SMC. A third advantage is that program modules of this kind can be shared by the SMM. This application handles some of the same datasets as the SMC. For this reason incorporating pre-existing modules with appropriate functionality is a useful way of avoiding duplication of effort.

The program module used to handle model design data is a good example of this kind of module. For each primitive in the model design language (described in Section 5.1.2.1) a storage structure is defined with module level scope. The storage structure consists of a counter variable to indicate the number of instances of the primitive in the model design and one or more arrays to hold information from each instance. For example, consider the information on state variables held in the model design. Five items of data are associated with each state-variable instance in the design: its name, the name of the modelled entity with which it is associated, a description, the units, maximum and minimum value that it can take. This information is held in five arrays. Elements in the each of the arrays of the same index correspond to a single state variable instance.

The program module also supports interrogation of the model design by other components of the SMM. As the model design data has module level scope it cannot be accessed directly by other SMM components. Instead different functions are called which return different sets of data. For example, state-variable information is returned by the VB procedure 'get_design_svs'. This procedure has one input (the name of the modelled entity for which state-variable information is required) and two outputs (a value giving the number of state-variables associated with the modelled entity given as input and an array containing the names of the state-variables). Further information (maximum and minimum values, description, units) on an individual state-variable can be returned using the procedure 'get_sv_details' using the name of the state-variable as an input.

The program module reads information from the Model design datastore when a new model is selected for use. Because the structure of information in this file is complex a reasonably sophisticated lexical analyser is used in the VB procedure which reads the information. Although the program module uses simple structures to represent the information in a model design quite a primitive way it is faster to implement and (because it is less complex) easier to maintain. This leads to faster prototyping and ultimately superior design. Furthermore the processing involved is not speed critical as the performance of the SMM is limited primarily by the time taken to execute simulation operations.

Table 6.1: Program modules used in the SMC application

Module name	Functionality
mc_control.bas	MC control
desc.bas	Creation of Model Descriptions.
pars.bas	Creation of Parameter Sets
code_gen.bas	Creation of Model Source-code
design.bas	Handling of Model Designs
mdule_des.bas	Handling of module designs
ini.bas	Handling of SMC initialisation data

6.2 SYMFOR MODEL MANAGER (SMM)

This application was created using VB. It implements seven of the processes from the SYMFOR DFD. Two of these are general-purpose processes (SID Maker and Run Controller) while six are display processes (Profile Viewer, Plan Viewer, Aggregate Information Tabulator, Individual Information Tabulator, Frequency Distribution Plotter and Time Series Plotter).

6.2.1 Interface

The SMM conforms to the MS Windows Multiple Document Interface (MDI) standard, as described in Petzold (1992). In this architecture there are three kinds of window:

- Parent window. There is only one of this kind of window in an application. This window possesses a toolbar, a menubar and a status box. It opens when the SMM application opens and closing the window terminates the application.

- Child windows. These occur within the client area of the parent window. They do not possess a menubar or toolbar of their own. They are modeless in that it is possible to interact with other windows of the application while the child window is open. These windows are typically used to display information or to facilitate user input which is not tied to a discrete activity

- Dialog-boxes. These boxes are not limited to the client area of the parent window. They do not possess a menubar or toolbar of their own. They are modal in that interaction with other windows of the application is blocked while they are open. These windows are typically used to either warn the user of something or solicit input from the user before an activity can proceed.

The windows may be divided into two kinds: control windows and display windows. The primary purpose of control windows is to allow users to manage and carry out SMM activities. The primary purpose of display windows is to present information. They do this by displaying information on modelled entities in the modelled stand. Two concepts are used to specify the characteristics of displays: *properties* and *instructions.* A display property is a scalar *i.e.* it can only have one value for a particular display. An example is the width of the modelled stand displayed in the Profile Viewer. A display instruction is a prescription concerning processing of modelled entity data that is used in the generation of the display. There can be more than one display instruction associated with a display. For example, an instruction might specify that individual trees of species group 3 should be displayed in green on the Profile Viewer.

There are 20 windows in total in the SMM Interface: 2 control windows, 7 display windows and 10 dialog-boxes. Table 6.2 lists the various windows used, and the rest of the section considers each in turn.

Table 6.2: Windows used in the SMM interface

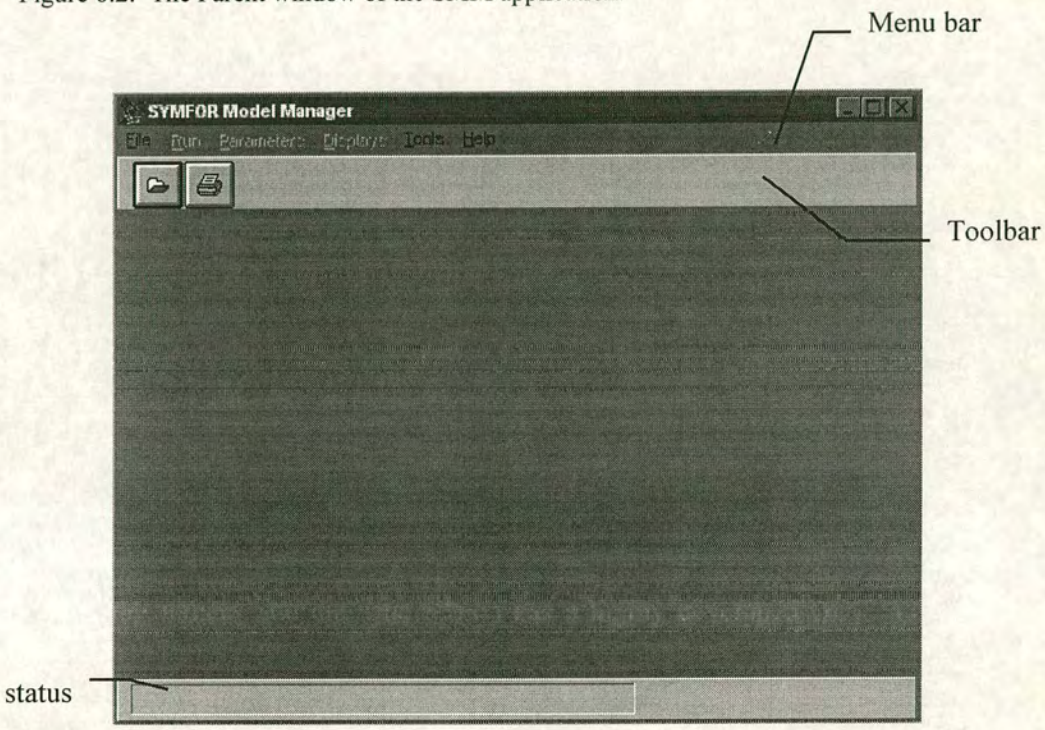
Window type	Window	Role
Control	Parent window	Manages all other windows
	Run control	Controls model simulation runs
Display	Model Description viewer	Displays natural language description of model
	Plan Viewer	Produces map showing locations of modelled entities
	Profile Viewer	Produces diagram showing the modelled stand in cross-section
	Aggregate Information Tabulator	Produces a table showing aggregate stand values such as total biomass
	Individual Information Tabulator	Produces a table showing the values associated with individual modelled entities
	Frequency Distribution Plotter	Produces a diagram showing the number of modelled entities in different categories
	Time Series Plotter	Graphs stand characteristics through time
Dialog-boxes	Open Model	Allows users to select a model for use in the SMM
	Parameter Editor	Allows users to change values associated with model parameters
	Parameter Disaggregator	Allows users to change values associated with disaggregated parameters
	SID Maker	Allows users to create new SIDs by combining Field Datasets
	Run Settings	Allows users to change values collected in individual runs and run length
	Run Initialisation	Allows users to specify the SID and parameter set used in an individual model run
	Criterion Selector	Allows users to specify a subset of modelled entities for use in display processing
	Cell Contents	Allows users to specify the contents of a cell in the Aggregate Information Tabulator
	Plane Viewer Options Panel	Allows users to specify the functioning of the Plan Viewer
	Profile Viewer Options Panel	Allows users to specify the functioning of the Profile Viewer
	FD Options Panel	Allows users to specify the functioning of the Frequency Distribution Plotter

Control windows

6.2.1.1 Parent window

The SMM Parent window is used to perform top-level activities such as opening child windows and printing child windows. The Parent window opens when the SMM application opens. Closing the Parent window closes the SMM application. Figure 6.2 shows the configuration of the SMM window.

Figure 6.2: The Parent window of the SMM application.



The user controls the SMM using menu-commands and toolbar-buttons. Every toolbar-button has a menu-command equivalent but not every menu-command can be performed using the tool-bar. Details of toolbar-buttons, menu commands and the actions they perform are given in Table 6.3.

The SMM has a number of different modes which are used to both enable and constrain user actions. In each configuration certain toolbar-buttons and menu-commands are available and others are disabled. The modes are:

① no model selected (the mode when software is first loaded);

② model selected, not running;

③ model selected and running;

The SMM moves between modes in reponse to user actions. Availability of toolbar-buttons and menu commands in each of these modes is indicated in Tables 6.3 and 6.4.

Table 6.3: SMM user actions and modes (I). Where a change in mode of the SMM results from a user action details of the transition are given. The availability of each action in different modes is indicated in the three 'Availability' columns - shading indicates the option is available.

Menu heading	Menu command	Toolbar icon	Action	Availability ①	②	③
File	Open model		Opens model ① - ②			
	Close model	-	Closes model ② ①			
	Describe model		Opens model description window			
	Print settings	-	Opens printer settings dialog			
	Print open displays		Prints open displays			
	Exit	-	Closes SMM ③ - ② - ①			
Run	Initialise new run		② - ③			
	Finish run	-	③ - ②			
	Clear previous run results	-	Deletes contents of results file			
	Change run settings	-	Opens run settings panel			
Parameters	Edit parameters		Opens parameter editor window			

Table 6.4: SMM user actions and modes (II). Where a change in mode of the SMM results from a user action details of the transition are given. The availability of each action in different modes is indicated in the three 'Availability' columns

Menu heading	Menu command	Action	Toolbar	Availability ①	②	③
Displays	Stand Table	Opens Stand Table display				
	Frequency Distribution	Opens Frequency Distribution display				
	Profile View	Opens Profile View display				
	Plan View	Opens Plan View display				
	Individual information	Opens Individual info. display				
	Time series plot current run	Opens TS Plotter display				
	Time series plot previous runs	Opens TS Plotter display				
	Copy Display	Copies active display to clipboard				
	Open Display options panel	Opens options panel for active display*				
Display arrangement	Cascade	Cascades child windows				
	Tile	Tiles child windows				
Tools	Make new Stand Initialisation File	Opens SID Maker window				
Help	About SYMFOR	Opens SYMFOR information*				

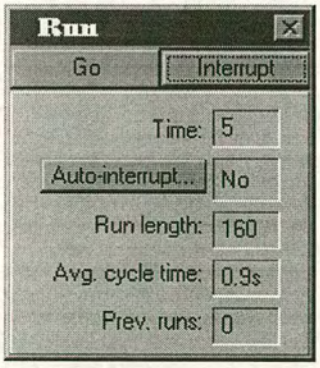
6.2.1.2 Run Control window

This window is used to monitor progress and speed and to control simulation runs of a model. It opens automatically if the user successfully uses the Run Initialisation dialog-box. The configuration of the window is shown in Figure 6.3. The information displayed on the window is:

- the number of years of the simulation run that have been completed;
- the maximum run length;
- the average time for a year's worth of calculations - this value is useful for assessing performance of the model on different computers;
- the next point at which the run should be interrupted (if defined).

The two push-buttons at the top of the window are used to start/ restart and to interrupt the run. Clicking on the push-button labelled 'auto-interrupt' and entering a value in the dialog-box that appears will set the next point at which a run should automatically be interrupted.

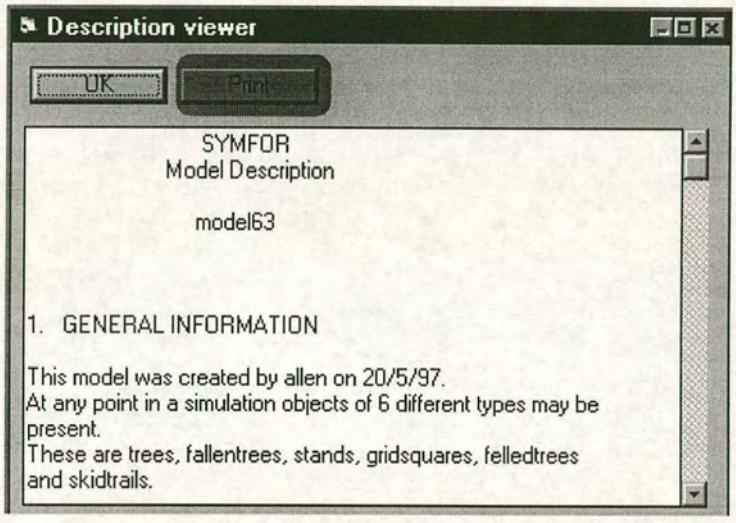
Figure 6.3: The Run Control window



6.2.1.3 Model Description Viewer

This window displays a natural language description of the model being used. The structure of the description is indicated in Section 5.1.2.5. The configuration of the window is shown in Figure 6.4. The window is opened using the 'Display model description' menu command under 'File' on the menu of the top-level window or by clicking on the equivalent tool-bar-button (Table 6.3).

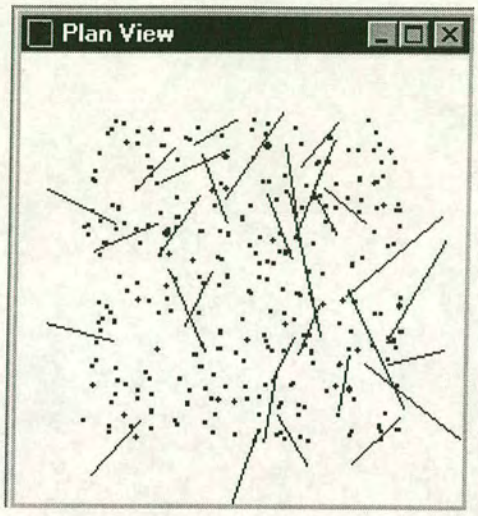
Figure 6.4: Model Description Viewer window



6.2.1.4 Plan Viewer

This window presents a diagram showing the locations and shapes of modelled entities within the modelled stand. Each modelled entity is represented by a point, straight line or shape. When the modelled entity is represented by a point then the size of the point within the display may scale according to an attribute possessed by the modelled entity. Different colours can be used to identify different sets of individual modelled entities. The window is shown in Figure 6.5, in which standing trees are represented by circles and fallen trees are represented by straight lines.

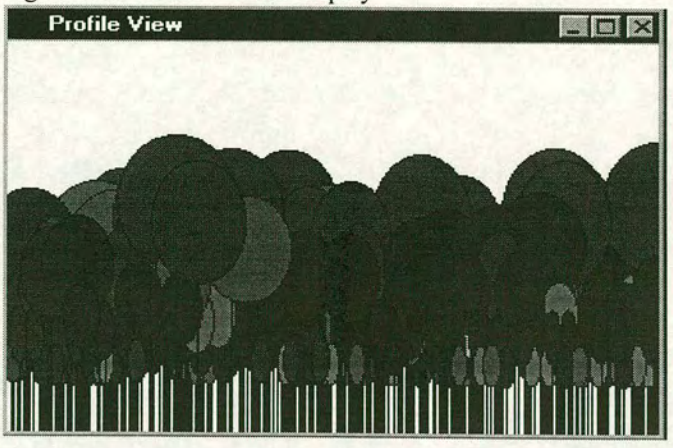
Figure 6.5: Plan Viewer display



6.2.1.5 Profile Viewer

This window presents a diagram showing the locations and shapes of modelled entities of type 'tree' within the modelled stand. Each individual tree is represented by an ellipse (the crown) and a line (the stem). The dimensions of the tree on the diagram are related to the tree attributes of total height, crown-point and crown-radius. If tree modelled entities are not defined in the Model Design or the definition does not include these three attributes then this display cannot be used. Different colours are used to identify different sets of individual modelled entities.

Figure 6.6: Profile Viewer display



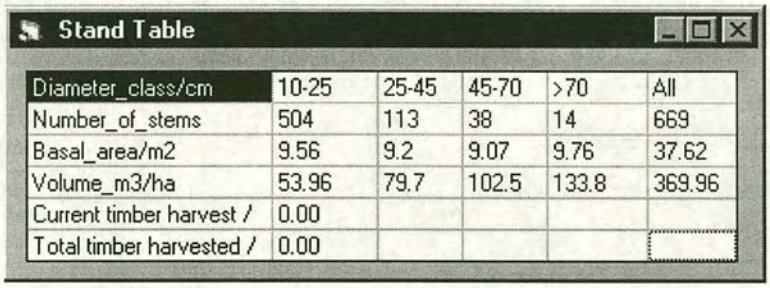
6.2.1.6 Aggregate Information Tabulator

This window displays a table of aggregate values from the stand. Each cell in the table contains either some text or a value. The window is shown in Figure 6.7. Aggregate values are values obtained by an operation on a set of individual modelled entities (the set is specified by the 'Criterion Selector' dialog-box (Section 6.2.1.14)). Three types of operation can be performed on such a set:

- counting the number of individuals in the set;
- summation of an attribute for all individuals in the set;
- calculation of the mean value of an attribute for all individuals in the set;

What appears in each cell is specified using the 'Cell Contents' dialog-box (Section 6.2.1.14). The number of cells in the table is specified by double-clicking on the border and entering the number of columns and rows that are required in the dialog-box.

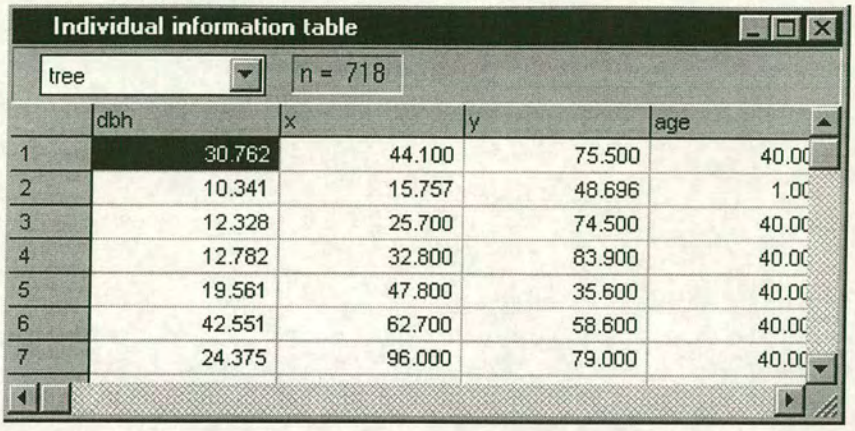
Figure 6.7: Aggregate Information Tabulator display



6.2.1.7 Individual Information Tabulator

This window presents a table that contains information on all individual modelled entities of a particular type (Figure 6.8). Each row corresponds to one modelled entities and columns correspond to attributes possessed by the individuals. A popup list box allows users to specify the object type used in the table. There is no Options Panel for this display.

Figure 6.8: Individual Information Tabulator

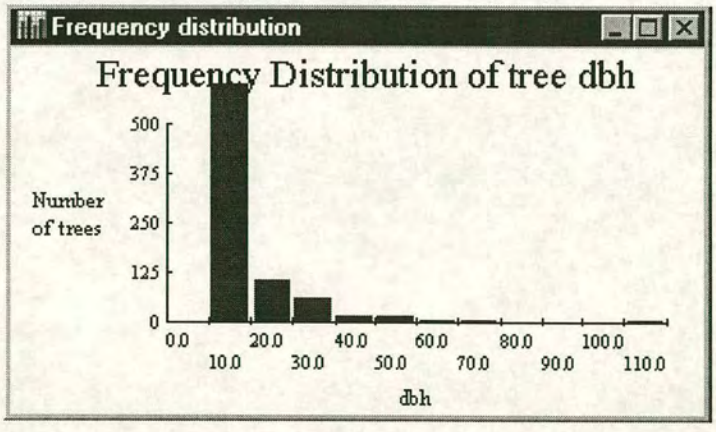


	dbh	x	y	age
1	30.762	44.100	75.500	40.0
2	10.341	15.757	48.696	1.0
3	12.328	25.700	74.500	40.0
4	12.782	32.800	83.900	40.0
5	19.561	47.800	35.600	40.0
6	42.551	62.700	58.600	40.0
7	24.375	96.000	79.000	40.0

6.2.1.8 Frequency Distribution Plotter

This window presents a diagram showing how many modelled entities of a particular type fall into each of a set of categories (*i.e.* a frequency distribution or FD). Each FD is for a single variable and uses a predefined set of classes.

Figure 6.9: Frequency Distribution Plotter

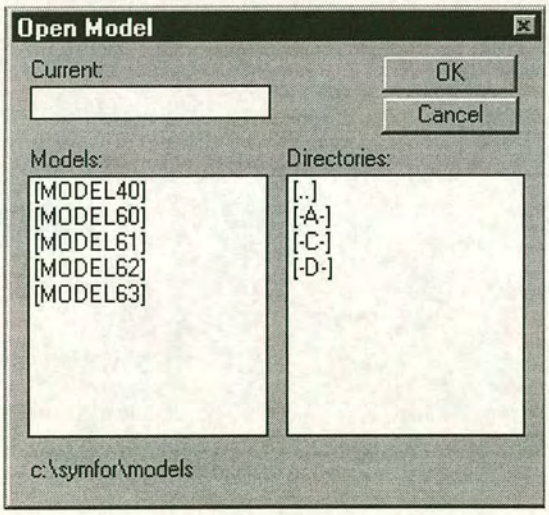


6.2.1.9 Open Model dialog-box

This dialog-box is used to choose a model for loading into the SMM. It is opened using the 'Open Model' menu command or the equivalent push-button on the toolbar. The configuration of the dialog-box is shown in Figure 6.10.

The dialog-box emulates the functionality of 'File open' dialog-boxes of MS Windows 3.x applications. Users navigate to the folder in which the model they want to load is located using the 'Directory' list-box. Double-clicking on a folder name in this listbox will open it resulting in the update of the 'Model' and 'Directories' listboxes and the path label (at the bottom left of the dialog-box).

Figure 6.10: Open Model dialog-box



6.2.1.10 Parameter Editor dialog-box

This dialog-box is used to change the value of model parameters. It is opened using the 'Edit parameters' menu command or the equivalent toolbar-button. The configuration of the window is shown in Figure 6.11.

The Parameters are grouped according to the SYMFOR module they belong to. The groups are contained in the list-box in the left hand corner of the dialog-box. When the user selects a new group the current parameters in the window are replaced by those that are associated with the new group.

There are two kinds of parameters: scalar and disaggregated. For both of these kinds the name, minimum and maximum and units are displayed in labels. If the user clicks on the name then a modal box containing a text description of the parameter will appear. In the case of a scalar parameter there is a text-box into which user can enter the new value. In the case of a disaggregated parameter there is a push-button labelled 'Disagg' where the text-box of a scalar parameter would be. When this push-button is pressed a dialog-box containing a set of values for the parameter appears (the Parameter Disaggregator dialog-box - Figure 6.12). The values appear in a grid and each can be edited by selecting a cell in the grid, entering a new value in the text-box at the top of the window and clicking on the push-button labelled with the tick.

Figure 6.11: Parameter Editor dialog-box

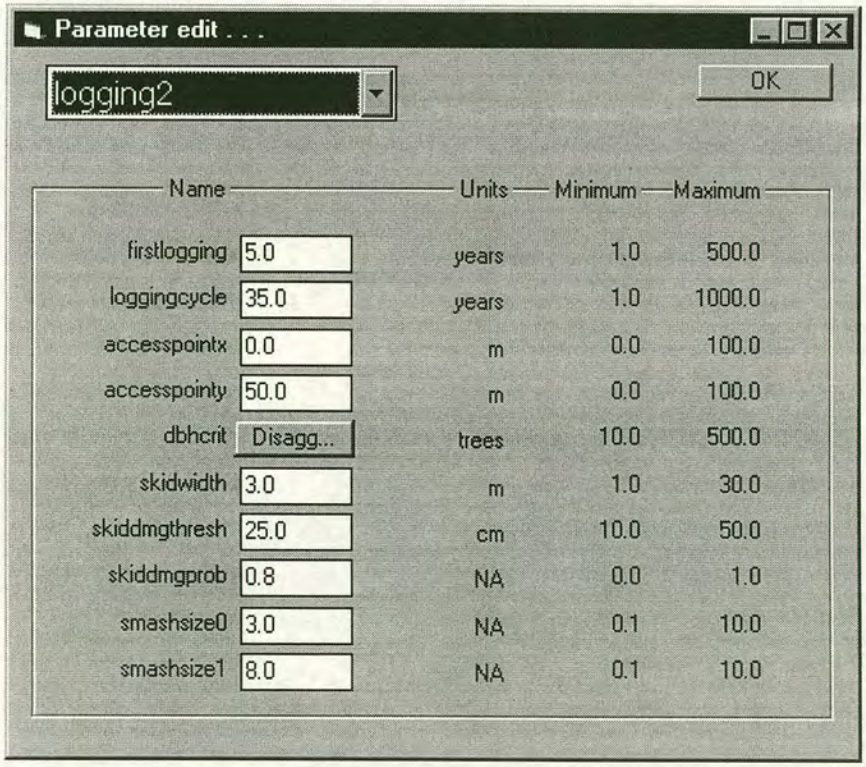
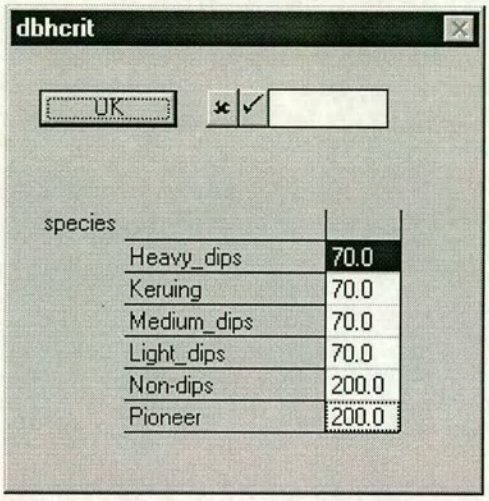


Figure 6.12: The Parameter Disaggregator dialog-box for a disaggregated parameter. Values for a variable (dbhcrit) which is disaggregated by species-group are shown



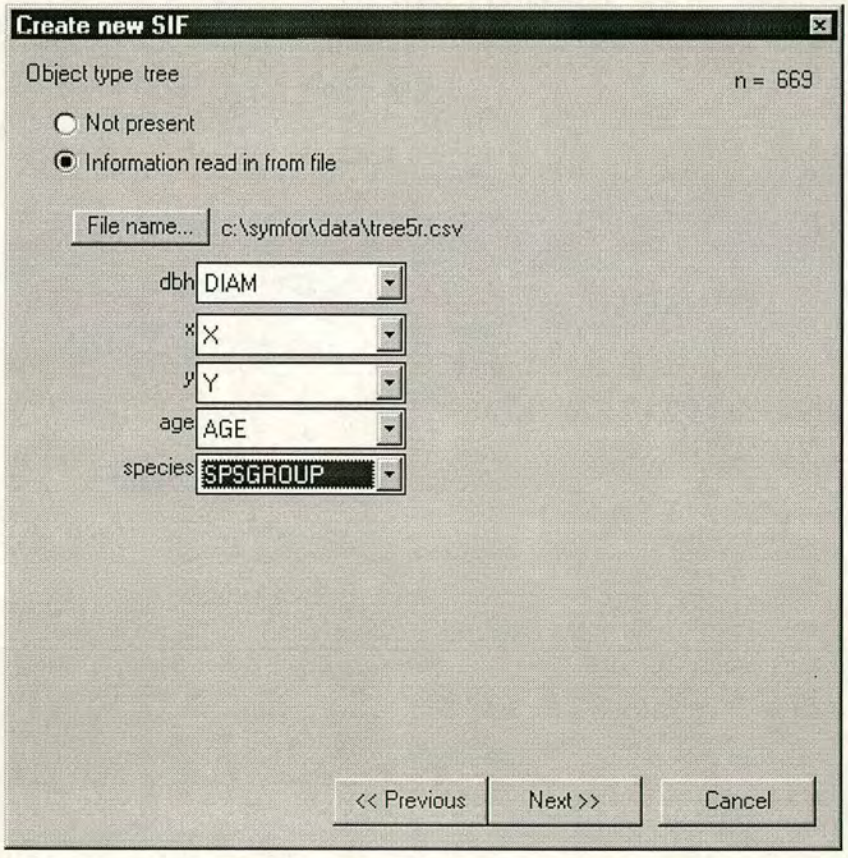
6.2.1.11 SID Maker window

This dialog-box is used to create Stand Initialisation Datasets (SIDs) compatible with the model that is in use. SIDs are used to specify the modelled entities present at the start of a simulation run. The algorithm used by the SID Maker is described in Section 5.1.3.7.

The SID Maker window works in a similar way to an Microsoft wizard *i.e.* by guiding users through a sequence of steps associated with a sequence of window configurations. The user completes a step by specifying values for all choices in a window. The user proceeds to the next step (*i.e.* a new window configuration) by clicking on 'Next' (and can go back to the previous step by clicking on 'Previous').

In the SID Maker each step requires users to specify initialisation details for a type of modelled entity that is present in the model design. Each window configuration allows the user to either specify that no individuals of the type are initially present or to specify how information from a field data file may be used to specify the modelled entities initially present. In the latter case the user is required to specify which fields from the data file are used to supply initial values for the model entity attributes. They do this by selecting the field to be used from a popup box that appears beside every attribute. Only fields that are suitable for use with the attribute appear. For example, a field containing text cannot be used to initialise state-variables. Once the user has finished the specification of an individual modelled entity type the 'Next' push-button allows them to progress to the next modelled entity type.

Figure 6.13: SID Maker window



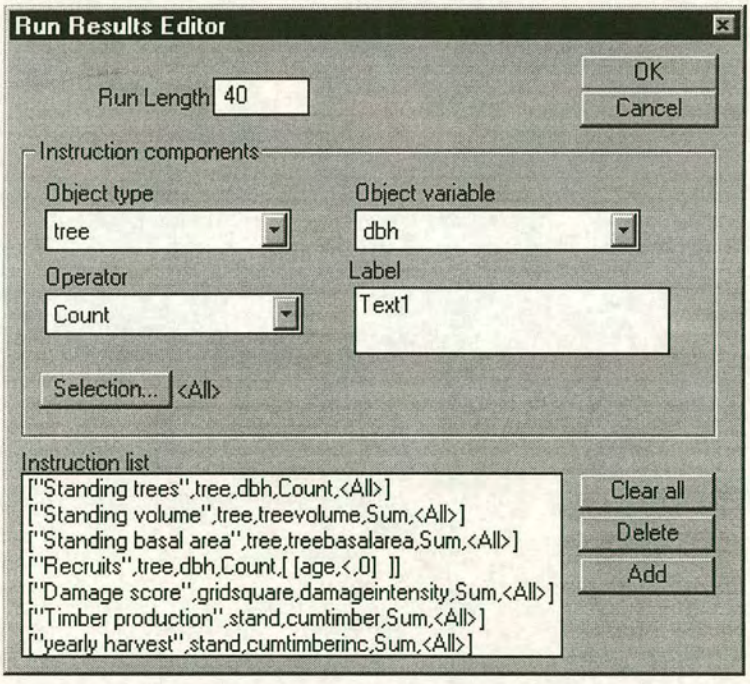
6.2.1.12 Run Settings dialog-box

This dialog-box is used to set certain characteristics that determine the nature of the simulation run. Specifically these characteristics are:

- the set of simulation results to be collected at the end of every cycle (henceforth abbreviated to simulation data);
- the maximum length of the run;

The configuration of the window is shown in Figure 6.14. It is opened by using the 'Run settings' menu command.

Figure 6.14: Run Settings dialog-box



A series of instructions are used to prescribe the simulation results that are collected. Each Instruction prescribes how a single datum should be calculated and named. Calculations involve an operation on a set of data belonging to individual modelled entities. Five controls are responsible for specifying instruction components:

- the 'Object type' popup box. This box contains all the modelled entity types in the model.
- the 'Object variable' popup box. This box contains all the variables associated with the currently selected object type
- the 'Operator' popup box. This box contains operators that can be used in Instructions. There are three of these: 'sum', 'mean' and 'count';
- the 'Selection' push-button. This push-button opens the 'Criterion Selector' dialog-box which allows users to select a set of modelled entities of the same type. When this has been done a text version of the selection appearing next to the 'Selection' push-button;
- the 'Label' textbox. This is used to specify the name given to the datum calculated by the instruction.

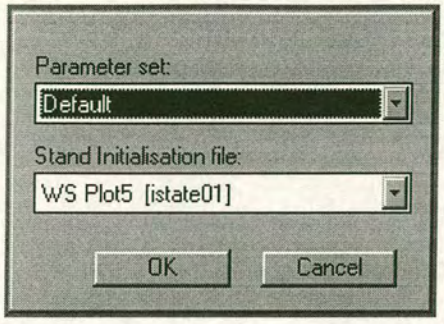
Once an individual instruction for a particular datum has been specified it can be placed in the 'Instruction list' box by clicking on the 'Add' push-button. The 'Delete' push-button can be used to remove selected instructions.

6.2.1.13 Run Initialisation dialog-box

This dialog-box is used to specify the parameter source and the Stand Initialisation File that are to be used in a simulation run. It is opened by the 'Initialise new run' menu command or the equivalent toolbar-button (Table 6.3).

The configuration of the dialog-box is shown in Figure 6.15. The 'Stand Initialisation File' popup box contains all the SIDs that are available for use with the model currently in use. The dialog-box is dismissed by the 'OK' push-button (in which case Run Initialisation proceeds to the next phase) or by the 'Cancel' push-button (in which case Run Initialisation fails).

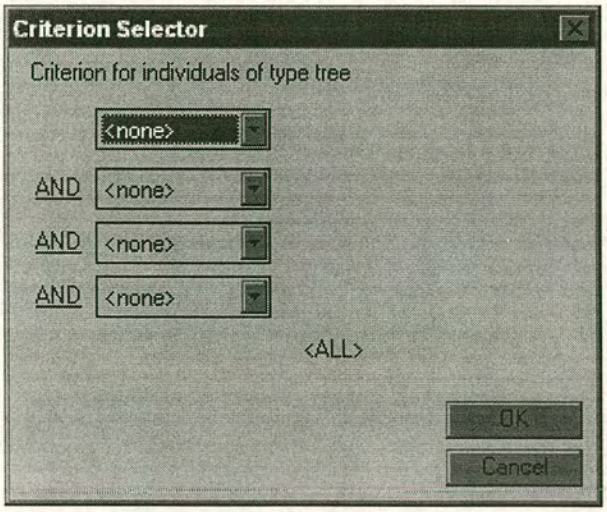
Figure 6.15: Run Initialisation dialog-box



6.2.1.14 Criterion Selector dialog-box

This dialog-box is used to specify a set of modelled entities (all of which must be of one type). It is launched by Display Options Panels in which there is a requirement to refer to a set of individuals *e.g.* the Plan View Options Panel. It works by allowing users to specify a logical expression consisting of up to four inequality terms joined by logical ANDs. The dialog-box is opened by clicking on one of the 'Selection' push-buttons that occur on display Options Panels and also on the Run Settings window. The configuration of the dialog-box is shown in Figure 6.16.

Figure 6.16: Criterion Selector dialog-box



The modelled entity type that the dialog works with is indicated in the top left of the dialog-box. (Note that this is specified by the user before the dialog-box is opened in the calling window). The box contains a number of popup list boxes. By default the item '<none>' is selected for each of the popup boxes. The other items in the popup list are attributes possessed by individuals of the modelled entity type specified for the dialog-box. When the user selects a different item from one of the popup boxes then the attribute selected becomes the first term in an inequality expression. Two boxes appear aligned horizontally with the original box: a popup list box containing a list of inequality operators and a text-box that allows users to complete the inequality expression.

6.2.1.15 Cell Contents dialog-box

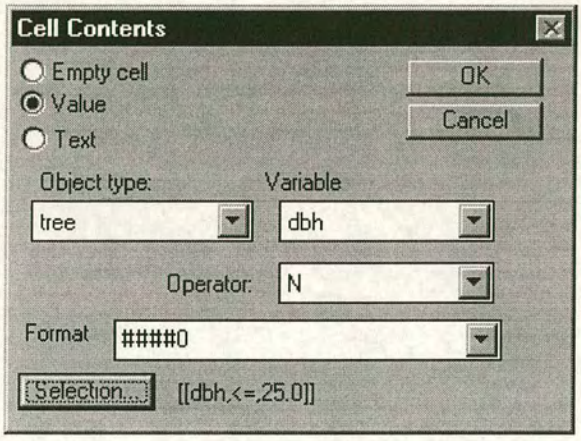
This dialog-box is used to specify what should appear in the individual cells in a table created by the Aggregate Information Tabulator. It is opened by double-clicking on the cell in an Aggregate Information table for which a new contents specification is required. After it opens the user can select one of three possibilities for display in the cell: nothing (*i.e.* leave the cell empty), text, or an aggregate value. The choice is made using the options in the top left. When the 'Empty cell' option is selected no additional features appear in the box. When the 'Text' option is selected a single text box appears in the box. Users type the text that they desire to appear in the cell into this box.

When the 'Value' option is selected controls appear that allow the user to specify how an aggregate value that will appear in the cell should be calculated. There are five of these controls:

- an 'Object' list box which allows users to choose the type of modelled entity that occurs in the set of individual modelled entities;
- a 'Variable' list-box which allows users to choose the modelled entity variable that will be used in the calculation of the aggregate value (note that when the type of the operation is a 'count', the contents of this list-box are irrelevant);
- an 'Operator' list-box which allows users to select the type of operation that will be used on the selection of individuals. Three operators can be used: sum, mean or count (N)).

- a 'Selection' push-button: this push-button opens the 'Criterion selector' dialog-box which enables users to choose the individual modelled entities to be used in the determination of the aggregate value;
- a 'Format' combo-box. This allows users to specify how the value will be displayed in terms of number of significant figures, notation etc. The user can choose one of the pre-existing formats or type in their own format expression.

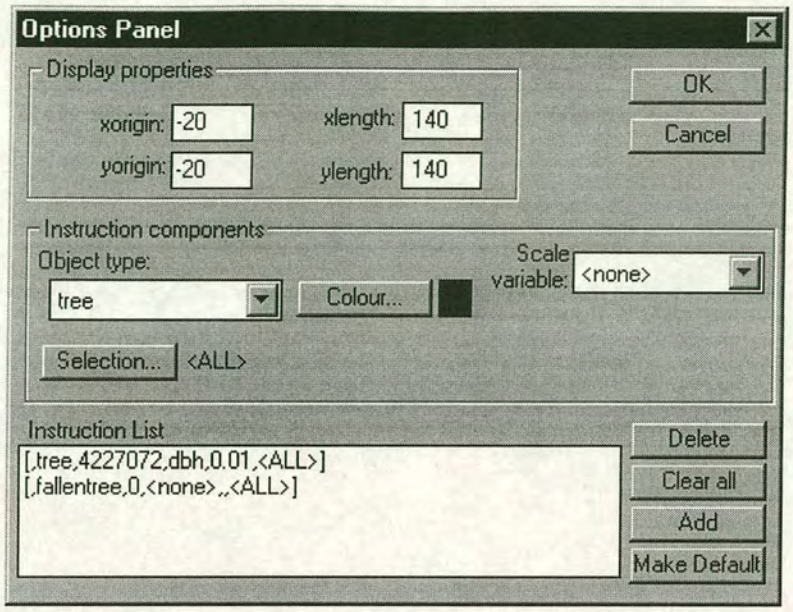
Figure 6.17: Cell Contents dialog-box



6.2.1.16 Plan Viewer Options Panel

This dialog-box is used to specify the characteristics of the Plan Viewer display. It is opened by double-clicking on the Plan Viewer display or by making the Plan Viewer the active display and selecting the menu command 'Open Display Options Panel'. The panel is shown in Figure 6.18.

Figure 6.18: Plan Viewer Options Panel



Display properties are used to specify the location and dimensions of the viewport (the area of the stand represented within the display) using the co-ordinate system of the modelled stand. Four controls are used:

- the 'x-origin' textbox - used to set the x-coordinate of the origin of the viewport.
- the 'y-origin' textbox - used to set the y-coordinate of the origin of the viewport.
- the 'x-length' textbox - used to set the length in the x direction of the viewport.
- the 'y-length' textbox - used to set the length in the y direction of the viewport.

Instructions are used to prescribe how different sets of individual modelled entities should be represented in the display. The controls used to specify instruction components are:

- an 'object type' popup listbox used to specify an modelled entity type. The object types listed will be all those specified in the model design that can be plotted *i.e.* all those that have attributes that bear interpretation as a point, line or shape;

- a 'Plot' popup listbox used to determine the symbols used in the representation of the modelled entities in the window.

- a 'Scale variable' popup listbox used to specify the variable that should be used to determine the size that an modelled entity represented by a point occupies on the display. This control is only visible if Plot is set to 'Point'.

- a 'Colour' push-button used to open a standard Windows dialog-box for selection of a colour;

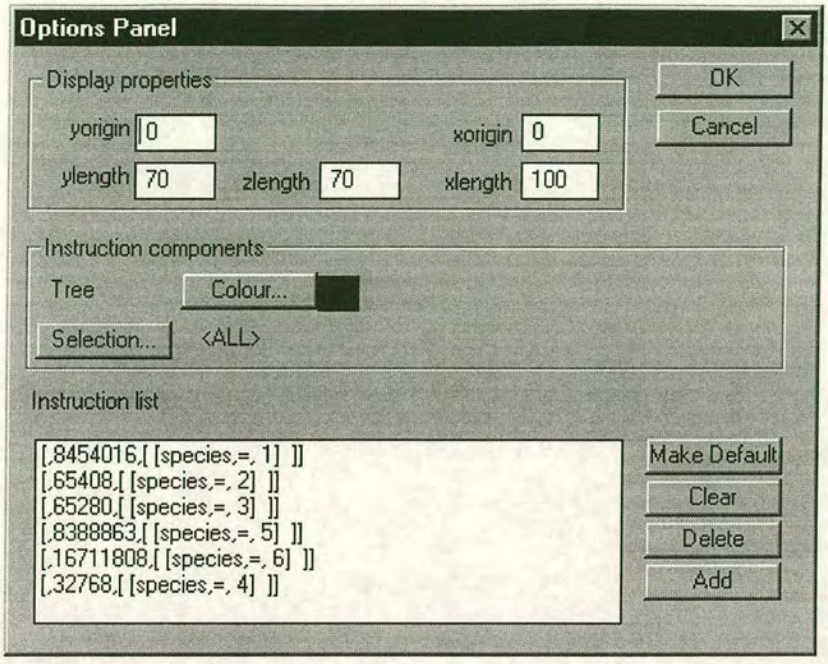
- a 'Selection' push-button used to open the Individual selector dialog-box to specify a set of individual modelled entities.

Once an individual instruction has been specified it can be placed in the 'Instruction list' box by clicking on the 'Add' push-button. The 'Delete' push-button can be used to remove selected instructions. Clicking on the 'OK' push-button the Plan Viewer closes the Options Panel. The Plan Viewer will then update to reflect any changes made. The new specification will be the one used every time the display is updated in the course of a simulation run.

6.2.1.17 Profile Viewer Options Panel

This dialog-box is used to specify the characteristics of the Profile Viewer display. It is opened by double-clicking on the Profile Viewer display or by making the Profile Viewer the active display and selecting the menu command 'Open display Options Panel'. Figure 6.19 shows the Options Panel.

Figure 6.19: Profile Viewer Options Panel



Display properties are used to specify the location and dimensions of the viewport (the area of the stand represented within the display) using the co-ordinate system of the modelled stand. Four controls are used:

- the 'x-origin' textbox - used to set the x-coordinate of the origin of the viewport.
- the 'y-origin' textbox - used to set the y-coordinate of the origin of the viewport.
- the 'x-length' textbox - used to set the length in the x direction of the viewport.
- the 'y-length' textbox - used to set the length in the y direction of the viewport.

Instructions prescribe how different sets of individual trees are represented on the 'Plan Viewer' display. The controls used to specify instruction components are:

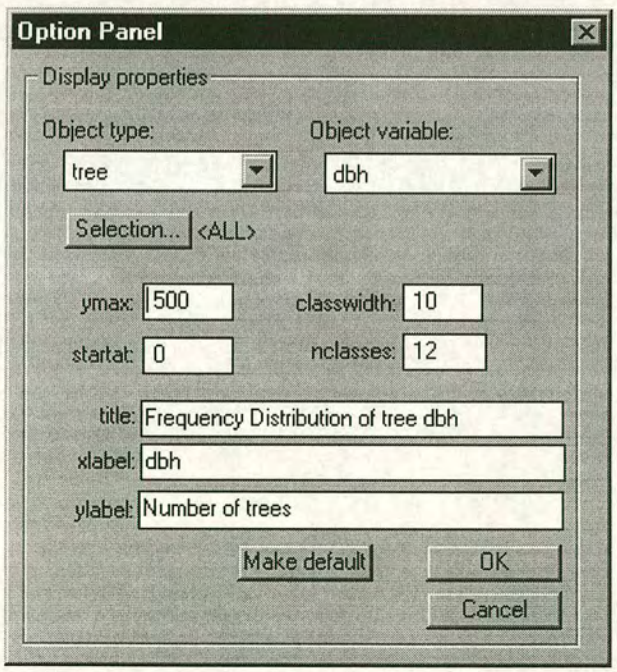
- a 'Colour' push-button used to open a standard Windows dialog-box for selection of a colour;
- a 'Selection' push-button used to open the 'Criterion Selector' dialog-box to specify a set of individual trees;

Once an individual instruction has been specified it can be placed on the 'Instruction list' box by clicking on the 'Add' push-button. The 'Delete' push-button can be used to remove selected instructions. When the Options Panel is closed by clicking on the 'OK' push-button the Profile Viewer will update to reflect any changes made. The new specification will be the one used every time the display is updated in the course of a simulation run.

6.2.1.18 Frequency Distribution Options Panel

This dialog-box is used to specify the characteristics of the FD Plotter. It is a modal dialog-box that is opened by double-clicking on the FD Plotter or by making the FD Plotter the active display and selecting the menu command 'Open display Options Panel'. The panel is shown in Figure 6.20.

Figure 6.20: Frequency Distribution Options Panel



The display possesses display properties. There are 10 display properties:

- the 'Object type' popup listbox used to specify the modelled entity type used in the FD. The object types listed will be all those specified in the model design that can be plotted *i.e.* all those that have attributes that can be interpreted as a point, line or shape;
- the 'Object variable' popup listbox used to specify the variable used in the distribution. The contents of the listbox depend upon the contents of the 'Object type' box;
- the 'ymax' textbox - used to set the upper value of y-axis;
- the 'startat' text box - used to set the lower value of the first class used in the distribution;
- the 'classwidth' text box - used to set the range of values in each class;
- the 'nclasses' text box - used to set the number of classes appearing on the x-axis of the distribution;
- the 'title' textbox - allows users to enter the title appearing on the FD;
- the 'x-label' text box used to specify the legend appearing on the x-axis of the FD;
- the 'y-label' text box used to specify the legend appearing on the y-axis of the FD;
- a 'Selection' push-button used to open the Individual selector dialog-box to specify the set of individual modelled entities used in the FD;

When the Options Panel is closed by clicking on the 'OK' push-button the FD will update to reflect any changes made. The new specification will be the one used every time the display is updated in the course of a simulation run.

6.2.2 Programming

The SMM application contains 13 program modules. One of these is responsible for control *i.e.* executing the transitions between SMM modes and managing model simulations. The control module has an associated integer variable called SMM state which stores the value of the current mode. It also has other variables that store information specified by the user in the course of using the SMM. For example, before entering state 2 the user must select a model: details of the choice are stored in a variable called 'selected model' associated with the module. All of the variables have module-level scope, *i.e.* they can be accessed by procedures in the same module but not procedures in forms or other modules in the application. The control module also contains procedures to handle changes in SMM state, one for each possible state transition (Table 6.5).

The control module is also responsible for controlling the model executable in simulation runs. It does this by establishing a Dynamic Data Exchange (DDE) conversation with the model executable. DDE is a protocol created by Microsoft for handling communication between different processes in the MS Windows operating system. The main advantage of using DDE is that it provides conventions for communication, so that there is no need to invent new ones.

Seven of the program modules are used to handle interaction with datasets. The approach of using program modules to handle interaction with datasets was described in Section 6.1.2. Four of these datasets are datastores in the SYMFOR DFD: Model Design, Module Database, Field Dataset, and TS Dataset. One is used to hold SID information (which gives the locations of files that are placed on a computer as part of SYMFOR installation), while the other dataset is the stand state vector, which occurs internally to the Model Executable.

Table 6.5: Actions taken on transition between SMM states.

Transition	Actions
$1 \rightarrow 2$	Model selection dialog-box displayed; User selects model; Model design loaded; Default model display settings read in; SMM reconfigured;
$2 \rightarrow 3$	Run initialisation dialog-box displayed; User selects SID and Parameter source; Run control window displayed; Model executable loaded into memory; DDE conversation between SMM and Model executable established; Model executable reads SID; SMM reconfigured;
$3 \rightarrow 2$	User warned about possible loss of data (if currently in the middle of a run); User prompted to save run results; Run control window unloaded; DDE conversation terminated and Model Executable terminated; SMM reconfigured;
$2 \rightarrow 1$	Specification cleared; Default settings from displays saved to file; SMM reconfigured;

Five are used to implement display functionality. Program modules are required because of problems transmitting data from the Display Options Panel to the Display. In VB version 3.0 it is not possible to call functions associated with forms from outside the form scope. This means that display data (*i.e.* display properties and instructions) has to reside in a program module, where it can be read and written by the display and the display options panel.

6.3 SYMFOR DLL

The SYMFOR DLL was created using MS Visual C++ version 1.5. Although it was created with a C++ compiler, it uses only C programming constructs. It does not contain any of the processes from the SYMFOR DFD (Figure 5.1). It can be therefore regarded as a feature of the physical design (and not the logical design) *i.e.* a feature that reflects constraints involved with implementation decisions rather than high-level conceptual design. It contributes functionality to Model Executable and SMM applications.

The SYMFOR DLL is used to perform SYMFOR tasks which are carried out in C and which need to be performed by more than one model executable. A DLL consists of a library of functions that may be used by an application without having to be compiled at the same time as the application. An important feature of DLLs is that they can be used by more than one application simultaneously. All of the functions in the DLL could conceivably be statically linked to individual model executables in SYMFOR. However dynamic linking of functions is preferable to static linking for a number of reasons:

- programs with statically linked functions are larger and take much longer to compile;
- when many programs share common statically linked functions the storage space required is larger compared to the situation if the functions were dynamically linking;
- if a function that is statically linked to multiple applications is changed then all programs containing the function must be re-compiled.

The SYMFOR DLL contains functions that:

- handle data-exchange between the SMM and individual SYMFOR model executables. - The SMM needs modelled entity information for use in displays and the model executables need to get the latest parameter and SIF information from the SMM;
- perform other common tasks. For example, random number generation is required by many models that include stochastic components. Dynamic linking means that code for this activity does not need to be included in every individual Model Executable.

The requirement for handling data exchange functionality within the DLL arises because of the different ways in which data is internally stored by model executables (C applications) and the SMM (a VB application). In the model executable, data on modelled entities is stored in such a way that all the data from a single modelled entity is held in consecutive memory locations. In the SMM modelled entity data is handled using a series of arrays. There is one array for each modelled entity attribute and array elements with the same index store data relating to the same modelled entity. In this case consecutive memory locations will contain data from the next modelled entity for the particular attribute. There is therefore a need to transform modelled entity data to a form that can be used by the model executable. While this could be separately performed by each model executable, as it is a common task it is more efficient to perform it in a DLL.

Table 6.6 lists the functions of the SYMFOR DLL. The functions are divided into four groups. Overhead functions are those which must always be present for the DLL to function in MS Windows. Modelled entity data-exchange functions are responsible for transforming modelled entity data from the model executable into a form accessible by VB applications. SIF data exchange functions are responsible for storing and making available the SIF currently in use. The parameter functions are used to store and access parameter values currently in use by a model. The geometry functions perform commonly used geometrical transformations and other geometry tasks. The general object handling functions perform common tasks for manipulating modelled entities.

Table 6.6: Funtions of the SYMFOR DLL

Grouping	Function name	Role
Overhead	LibMain	Entry procedure
	WEP	Exit procedure
modelled entity data exchange	StoreHandle	Accepts and stores Windows memory object handle
	FlushHandle	Clears current memory object handles
	GetNcases	Returns the number of modelled entities of a particular type
	GetIntVector	Returns an integer array of modelled entity attributes
	GetFloatVector	Returns a float array of modelled entity attributes
SID data exchange	RegisterSID	Accepts and stores the name of an SID
	ReturnSID	Returns the name of an SID
Parameter data exchange	RegisterParameterName	Accepts and stores a parameter name
	ParChange	Accepts and stores parameter values associated with a parameter name
	ParInit	Returns values associated with a parameter name
	ClearParameters	Clears currently registered parameter set
Geometry	MakeShape	Transforms set of Polar co-ordinates to Cartesian
	InsideShape	Determines if a point is inside a given shape
	ThickenLine	Generates co-ordinates of rectangle given axis and width
	GetExp	
General object handling	UseObject	Returns a memory pointer to an object given its handle
	RequestDynamicHandels	Allocates unused handles for objects which change in number in a simulation run
	RequestHandles	Allocates unused handles for objects which do not change in number

6.4 DISCUSSION

The implementation described in this chapter is closely related to the design discussed in the last chapter. Each of the processes occurs in one of two components: the SMC and the SMM (the only exception is the Model Executable of the design). There is however one 'component' of the implementation which does not have a direct precursor in the design, the SYMFOR DLL. The reasons for adopting the DLL are discussed in Section 6.3. They are mainly technical in nature. The process of adding in components to satisfy technical requirements not immediately obvious in the design is however a common feature of software design methodologies (*e.g.* Kendall and Kendall, 1992).

Two features distinguish the implementation or physical design of SYMFOR: strong support for visualisation of data and the use of more than one development system. SYMFOR features no less than 5 ways of displaying information on the state of the modelled stand at any point in a simulation, and the user to a very high extent can customise each of these. This approach is useful for two main reasons. First, it helps stimulate feedback from stakeholders. Design of SYMFOR proceeded by the production of many prototype versions. Stakeholders were exposed to each of these and asked to respond. The tools for visualisation were found to be critically important for stimulating comments from these stakeholders.

The second reason that flexible tools for visualisation are important is that they can increase the number of potential applications for the software. Different simulation systems and even different reports on silvicultural trials may present information in different formats or use different standards. One common problem with comparisons currently is that often different size-classes are used when creating stand tables. Another problem is that that different ways of presenting information are appropriate for investigating different objectives. For example, profile diagrams may be important for investigations into biodiversity, but irrelevant in investigations into timber production. Bell and O'Keefe (1987) and Rooks (1993) discuss how visualisation and interactive simulation can contribute to design quality of simulation software.

SYMFOR is also unusual (at least with respect to forest simulation systems) because two software development systems were used in its implementation (VB and VC). This approach means that SYMFOR is able to benefit from the particular strengths of each development system, while not being limited by the weaknesses of either.

VB is good for Rapid Application Development (RAD) but is computationally inefficient (as it is interpreted rather than compiled). The features that make it good at RAD are that the language it is based on (BASIC) is designed to be easy to use and that it supplies a series of visual components such as text-boxes, list-boxes that can readily be incorporated into applications. These features mean that the interface can be rapidly altered in response to feedback from stakeholders. As discussed previously, SYMFOR development was to a large extent driven by this feedback, and this approach undoubtedly contributed to the success of the system.

VC, in contrast, is not well suited for RAD, but is computationally efficient in that it is a compiled rather than interpreted language. It is also terse and versatile. These features mean that it is suitable for implementing functionality that is speed-critical. In SYMFOR this corresponds to the code of the Model Executable, which is responsible for performing all the functionality associated with model content e.g. calculation of rates of change, recalculating intermediate variables, updating the stand *etc*.

The main disadvantage of developing parts of the system using different systems is that facilitating their interoperation is more technically involved. For example, they must agree on the representation of data passed between them and the order in which data items are passed. Adopting a certain set of conventions can however solve most of these problems simply. In the case of SYMFOR MS Windows conventions were adopted.

Chapter 3 has considered the requirements for SYMFOR while Chapters 4-6 have considered the design of SYMFOR in relation to these requirements. While it has been argued that the design does indeed meet the requirements, most of this is argumentation has been fairly abstract or theoretical. There is therefore a need to consider the utility of SYMFOR in addressing a well-defined problem related to tropical forest dynamics, and it is this that is attempted in the next chapter.

7. Scenario Analysis using SYMFOR:

GROWTH AND YIELD PROJECTION IN INDONESIAN FORESTS

The purpose of this chapter is to demonstrate how an investigation with the potential to contribute sustainable management of tropical forests can be tackled using SYMFOR. The investigation was chosen with reference to the context of SYMFOR development *i.e.* as part a development programme with the goal of improving management of forests in Indonesia. For this reason an investigation into growth and yield of a forest in Kalimantan (Indonesian Borneo) under different felling cycles was undertaken.

The material covered in this chapter is important because it illustrates a number of important features of the framework. In particular it illustrates:

- Use of permanent sample plot data for growth and yield modelling. This is Requirement 5 of those listed in Chapter 3. It is particularly important given that one activity undertaken by the ITFMP was the establishment of a field station and peremanent sample plots in Central Kalimantan. The field station is named Wanariset Sangai and has 15 x 1 hectare permanent sample plots that have been established in lowland dipterocarp forest (Proctor, 1994). There is a need to show that data from these plots can be accommodated within the framework and used to make estimates of growth and yield.

- Application of an individual-based simulation model to growth and yield modelling. While individual-based models are common in the ecological literature, they are as yet rarely applied specifically to modelling change resulting from management operations (exceptions are Kohler and Huth, 1998; Kurpick *et al.*, 1997). There is therefore a need to reinforce the message that individual-based models are suitable for this purpose.

- Capture of processes occuring in real forest stands. There is a need to demonstrate that the model is sound in that it accurately captures the internal functioning of the system being modelled (Requirements 6-9; Bossel, 1994; Oderwald and Hans, 1993).

- Use of SYMFOR concepts (described in the ontology in Chapter 4) and the formal Model Design language for specifying a model. There is a need to demonstrate that the content of a model appropriate for conducting a serious investigation can be specified using the concepts (such as 'modelled entity', 'state-variable' *etc*) and mechanisms (*e.g.* production of a formal Model Design) provided in SYMFOR.

- Use of a simulation experiment with treatments and replication to evaluate alternative silvicultural systems. In addition to the content of a model, the use of a model is critical in establishing the superiority or otherwise of silvicultural systems. There is a need to demonstrate how a framework model can be used in an evaluation of alternative silvicultural treatments.

While the model in the case study is non-trivial (in that a model with a comparable level of complexity could be used to produce serious estimates), estimates of Growth and Yield produced by the case study are highly preliminary. This is so because of two main reasons. First there is still uncertainty in parameter values employed in the model. While the model is deliberately formulated so that all parameters can be obtained with a level of effort comparable to that currently employed, not all existing data sets contain appropriate measurements. While new protocols were developed for some of the new data required (Clearwater, 1996) there was not sufficient time to gather enough of this new data to support analysis. Second, there may be problems with model formulation that lead it to produce biased estimates. Even models that are correctly parameterised can contain elements which lead to pathological behaviour. Alder (1995) provides an example of how a polynomial equation can provide a good fit to a set of data, but can produce seriously flawed estimates when extrapolated even slightly. Some of the problems with bias can be offset by increasing the detail of the forest representation used. For example, inclusion of plot topography may mean that the higher growth rates of trees often observed on ridges (Whitmore, 1984) can be modelled so eliminating bias arising from omission of this feature in a model. Section 7.7.5 discusses this further.

The fact that current estimates produced by the framework may be unreliable does not detract from the design of SYMFOR. The most critical factor for long-term improved management is the participation of Indonesia-based forestry professionals. While the early production of reliable estimates may aid this several factors such as the ease-of-use of the framework, the support for visualisation and the adaptation of the framework in line with feedback received from potential users are much more critical.

7.1 INTRODUCTION TO INVESTIGATION

Length of felling cycle is one of the most important factors under control of the forest manager working with selection systems. In the Indonesian system of TPTI (Anon, 1992) the felling cycle is set at 35 years. In this system it is assumed that after a logging the trees forming the next crop are already present. This allows a relatively short time between fellings. There are a number of requirements for such a system to be successful including:

- Trees present in the stand that will make up the next crop (advance growth) must be sufficient in number and of the right species mix before logging takes place;
- Advance growth must remain intact throughout logging operations;
- Sufficient advance growth must grow to become loggable in the next cycle;
- Regeneration must be of the right species mix and sufficient in extent to ensure that third and later cycle yields are maintained.

Spatial variability in forest may confound attempts to manage large areas of forest sustainably using a single silvicultural system. However even small scale variability obtained over a few km^2, can lead to substantial variation in performance of silivicultural systems. It is important that such small-scale variability receives attention when designing experiments or extrapolating the results from model or empirical studies to formulate management guidelines. In a modelling approach this can be done by ensuring that a range of stand structures are considered when evaluating a particular silvicultural system.

7.1.1 Aim

To investigate the impact of different lengths of felling cycle on growth and yield of mixed dipterocarp forest.

7.2 MODEL DESIGN

The model captures the dynamics of a one hectare patch of lowland dipterocarp forest. The time-step used in model simulations was one year. Seven different types of modelled entity (ME) are captured (Table 7.1).

Tree, fallen-tree, felled-tree and cohort MEs are classified by species-group. Six species groups are used in the model (Table 7.2). Tree MEs are also classified by size. The size-classes used are 10-20 cm dbh, 20-30cm dbh, 30-50 cm dbh, 50-70 cm dbh and > 70 cm dbh. The classification-groups that individual MEs belong to determine the parameter values used in model calculations.

7.2.1 Trees

Each tree ME has the state-variable of stem diameter. The x and y co-ordinate of the tree are ME constants, while the species group to which the tree belongs is a constant classifier and the size-class of the tree is an intermediate classifier. There are intermediate variables of height, crown point, crown radius, basal area, volume, shadeindex and stem diameter increment. They are destroyed in logging events associated with the stand ME and in natural disturbance events associated with individual trees.

Table 7.1: The ME types that are included in the model design used in this investigation.

ME type	Referent	Rationale for using the ME type
Tree	A standing individual tree that is greater than 10 cm in stem diameter.	Modelling individuals means that local ecological interactions can be captured, and these are important in forest dynamics; modelling individuals avoids aggregation error.
Gridsquare	A square area of forest of 10 x 10 m	Gridsquares disaggregate the stand spatially. This assists the model in handling phenomena for which decomposition into natural individual MEs is not possible or useful.
Cohort	A group of seedlings or saplings less than 10 cm in stem diameter of a particular species-group that established at the same time within a gridsquare.	Individual-based submodel requires regeneration input; it is not practicable or useful to distinguish individual seedlings or saplings; use of cohorts avoids some of the problems associated with size-classes.
Fallen tree	A tree that has fallen within the stand	Fallen trees create stand disturbance that it is important to capture; fallen trees have different attributes to standing trees (*e.g.* damage zone) so that a separate ME type is required.
Felled tree	A tree that is selected and felled and forms part of the harvest obtained in harvesting operations	Felled trees have attributes that fallen trees do not have (*e.g.* merchantable volume and value) so that a separate ME type must be used.
Skidtrail	A track created by a tractor when it is used to remove a felled tree from the stand	Skidding creates disturbance within the stand; it is important to capture this disturbance; disturbance is localised within individual skidtrails.
Stand	A square area of forest of 100 m x 100 m	Some model operations must occur at a high organisational level *e.g.* summing the attributes of individual MEs or implementing a logging.

Table 7.2: Species-groups used in model simulations. Six different species-groups are used in the model. Note that some congeneric species occur in different groups - *Shorea* and *Hopea* both contribute to more than one group. Timber groups are given in brackets where appropriate

Abbreviation	Charateristics	Genera	Rationale
Heavy Dips	Slow growing; growth unresponsive to increased light; medium sized trees; long-lived;	*Shorea* (balau) *Hopea* (giam) *Vatica* (resak) *Cotylelobium* (resak)	Growth characteristics different from other dipterocarps; Grouping is the same as the commercial grouping of 'Heavy Hardwoods' recognised by foresters
Keruing	Medium growth rates;	*Dipterocarpus* (keruing)	Growth characteristics intermediate for dipterocarps; Regeneration characteristics different than from other Medium dipterocarps; Grouping recognised by foresters;
Medium Dips		*Hopea* (merawan) *Dryobalanops* (kapur)	Growth characteristics intermediate for dipterocarps;
Light Dips	Large trees; fast growing; long-lived;	*Shorea* (meranti) *Parashorea* (white seraya) *Anisoptera* (mersawa)	
Non Dips	Various	Non-dipterocarps that are not pioneers e.g.	Many species in this group have not been well enough studied to allow meaningful distinctions to be drawn;
Pioneers	Aggressive growth response to increased light; Relatively short-lived; Short in stature;	*Macaranga*; *Mallotus*; *Anthocephalus*;	Non-dipterocarps with obvious pioneer tendencies;

Total height is a function of tree diameter. The relationship has the form of a non-rectangular hyperbola and is given in Equation 7-1.

$$H_{total} = \frac{e \cdot D_{stem}}{1 + \left(\frac{e \cdot D_{stem}}{o} \right)}$$ (Equation 7.1)

where H_{total} is the total height of the tree in m, D_{stem} is the stem diameter of the tree in cm, e is a coefficient that controls the initial slope of the response curve, o is the maximum height that the tree can attain. The values of e and o used are disaggregated by species-group and are given in Table 7.5. The height of the crown is calculated using a simple linear function with the total height of the tree as the independent variable (Equation 7.2).

$$H_{crown} = H_{total} \cdot a$$ (Equation 7.2)

where H_{crown} is the height at which the first foliage of the tree occurs and a is a coefficient that controls the slope of the response. a is disaggregated by species-group, the values used being given in Table 7.5.

Crown diameter is linearly related to stem diameter (Equation 7.3). This form has been found to be appropriate for many species of tropical tree (Dawkins, 1963).

$$D_{crown} = b \cdot D_{stem}$$ (Equation 7.3)

where D_{crown} is the diameter of the crown in m and e is a coefficient that controls the slope of the response.

The volume equation used is given in Equation 7.4.

$$V = \frac{f \cdot H_{crown} \cdot \pi \cdot D_{stem}^{\ 2}}{4}$$ (Equation 7.4)

where V is the merchantable volume of the tree in m^3 and f is a form-factor the value of which is given in Table 7.4.

The relationship used to calculate basal area assumes that the stem cross section at the point of measurement is a perfect circle (Equation 7.5).

$$A_{stem} = \frac{\pi \cdot D_{stem}^{\ 2}}{4} \qquad \text{(Equation 7.5)}$$

where A_{stem} is the basal area of the tree in cm^2

The amount of foliage that a tree has is related to tree diameter. The relationship is a non-rectangular hyperbola (Equation 7.6).

$$A_{leaf} = \frac{A_{stem} \cdot u}{1 + \left(\frac{A_{stem} \cdot u}{v} \right)} \qquad \text{(Equation 7.6)}$$

where A_{leaf} is the leaf area of the tree in m^2, u is a coefficient giving the initial slope of the curve and v is the maximum amount of leaf area that an individual tree can have.

The method used to derive the shading index for a tree involves calculating the value of *distance weighted size ratio (*DWSR*)* for each neighbour of the tree. The definition of *neighbour* used in this model is a tree that occurs inside a circle with radius r_{nbr} and centred on the tree for which it is desired to calculate the value of shading index. The value of rnbris given in Table 7.4. The value of DWSR is found by first dividing the stem diameter of the neighbour by the stem diameter of the tree then dividing the result by the distance between the two trees. The value of shading index is given by the sum of the values of the distance weighted size ratio for all the neighbours. This variant of shading index is known as Hegyi's index (Dale, Doyle and Shugart, 1985) and can be formulated as in Equation 7.7.

$$I = \sum_{j=nnbrs}^{j=1} \frac{D_{stem(j)}}{D_{stem(i)}} \cdot \frac{1}{s_{(i,j)}} \qquad \text{(Equation7.7)}$$

where I is the value of shadeindex for the tree i is the tree for which the value of shading index is to be calculated, j is a neighbour of i, $s_{(i,j)}$ is the distance between tree i and tree j in m, *nnbrs* is the number of neighbours possessed by tree i

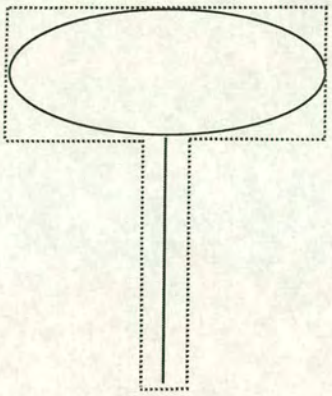
The relationship used to predict diameter increment of individual trees based on their shading index is given in Equation 7.8.

$$\Delta D_{stem} = k + 10^{-q \cdot \log_{10}(I)} \qquad \text{(Equation 7.8)}$$

where ΔD_{stem} is the stem diameter increment of the tree in cm yr^{-1}, k is a coefficient that controls the asymptotic value of ΔD_{stem}, l is a coefficient that determines the sensitivity of ΔD_{stem} to increasing values of I. A double logarithmic relationship of this kind is useful for minimising problems with heteroscedasticity of data when calibrating the relationship (Alder, 1995).

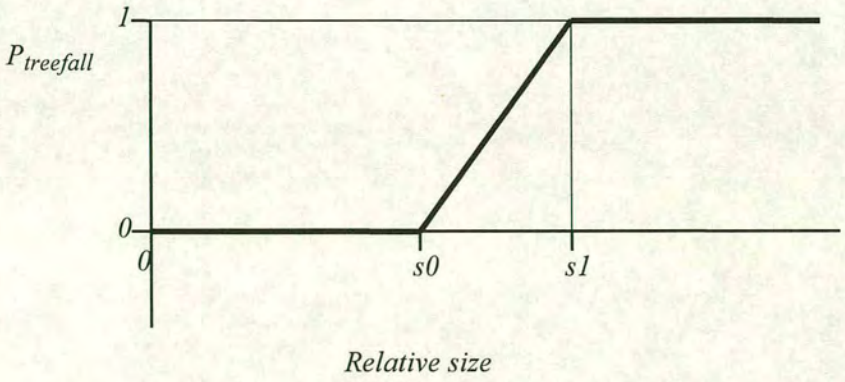
Tree MEs have an event called natural disturbance. The probability that an individual tree will become an initiator for a disturbance event in a year is assumed to be constant. The procedure for determining the secondary treefalls in the event is more complex. The shape reflecting the dimensions of the initiator is projected onto the plane of the ground surface of the stand. This shape is known as a *potential damage zone* (PDZ). Figure 7.1 illustrates how a PDZ is derived.

Figure 7.1: Plan view showing the relationship between potential damage zone dimensions and dimensions of falling tree. The damage zone consists of two rectangles, one associated with the crown of the tree and one associated its stem.



Each tree that occurs within a potential damage zone has a probability of undergoing secondary treefall. This probability is related to the relative size of the tree (the stem diameter of the falling tree divided by the stem diameter of the tree within the PDZ). The relationship is assumed to have three linear phases as shown in Figure 7.2, and is described using the two parameters s_0 and s_1.

Figure7.2: The relationship between $P_{treefall}$ and relative size



7.2.2 Gridsquare

Gridsquare MEs have three intermediate variables: total leaf area, LAI (leaf area index) and disturbance index. The total leaf area is calculated by summing all the foliage of trees within the gridsquare (Equation 7.9)

$$T = \sum_{i=ntrees}^{i=1} A_{leaf\,i} \qquad \text{(Equation 7.9)}$$

The value of LAI is calculated by taking the value of T and adding a factor to it that reflects the contribution of adjacent gridsquares to the LAI. An adjacent gridsquare is one of the eight surrounding gridsquares. The total contribution of the adjacent squares is obtained by summing the contribution of each adjacent. The foliage of the adjacent square is multiplied by a weighting factor to give its contribution. This is summarised in Equation 7.10.

$$l_i = T_i + \sum T_j \cdot w \qquad \text{(Equation 7.10)}$$

where l_i is the leaf area index of the gridsquare, i is the gridsquare for which LAI is to be calculated, j denotes a gridsquare adjacent to the gridsquare for which LAI is to be calculated and Tx is the total foliage in a gridsquare

A Monte Carlo method is employed for estimating gridsquare disturbance index. First a series of randomly located points within the gridsquare for which disturbance index is to be calculated are generated (the number of points is given by the parameter N (Table 7.4)). Second the number of these points that occur within one or more PDZs is determined. PDZs are associated with fallen-tree MEs, with felled-tree MEs and with skidtrail MEs. The value of disturbance index is then the number of points inside one or more PDZs divided by the total number of sample points within the gridsquare.

7.2.3 Cohort

Cohorts MEs have only one rate-of-change variable, that of stage increment and one state-variable, that of stage. There is one cohort of each species-group for every gridsquare ME. While the cohort ME persists, it may have periods when it is 'not active' which correspond to times when seedlings or saplings are not actually present in the gridsquare.

The value of stage increment is used to change the value of the cohort stage. The value of stage-increment depends upon two values of the gridsquare with which the cohort is associated: disturbance index and LAI. Stage-increment is either 1 or 0 or equal in magnitude and opposite in sign to stage. The conditions under which each of these values are obtained are shown in Table 7.3. However, stage increment is also used to *reset* cohorts. When this event happens the value of stage increment is set equal in magnitude to the value of the cohort stage. When the stage of the cohort reaches a critical value (given by the parameter $T_{maturity}$) the cohort creates new tree MEs of the same species-group as the cohort. The number of new of individuals produced by each cohort is governed by the parameter E. Values for both these parameters are given in Table 7.5.

Table 7.3: Calculation of stage increment associated with cohorts. D_{crit}, l_0 and l_1 are all model parameters. The value of D_{crit} is given in Table 7.4, while the values of l_0 and l_1 are given in Table 7.5

Condition	stage increment
gridsquare disturbance index < D_{crit} **AND** gridsquare LAI > l_0 **AND** gridsquare LAI < l_1	1
gridsquare LAI < l_0	0
gridsquare LAI > l_1	0
gridsquare disturbance index > D_{crit}	-stage

7.2.4 Fallen trees

Fallen-tree MEs are created by natural disturbance events associated with individual trees and by logging operations that are associated with the stand ME. In all cases of fallen tree creation there is an associated destruction of a tree ME.

Fallen-tree MEs have ME constants of dbh, volume and basal-area, an associated event called disintegration, a state-variable called lifetime and a rate-of-change variable called life-year. In addition they have a set of ME constants vx0, vx1 ... vx8 and vy0,vy1, ...vy8. These are used to store vertex co-ordinates of the potential damage zone shape that is associated with each fallen-tree ME. They also have the ME-constants x0, y0, x1 and y1 to store the co-ordinates of the base of the stem and the top of the stem.

Values for all the ME-constants (by definition) are set at the time when a fallen-tree ME is created and do not change through-out its lifetime. Some ME constants (dbh, volume and basal-area) are initialised directly with values from the destroyed tree ME attributes. Other ME constants (PDZ vertices, orientation and stem co-ordinates) are initialised with values obtained through transformation of the destroyed tree ME data in the destruction/creation event.

The value of the lifetime state-variable is initialised to zero. Every year it is incremented by the value of lifeyear (which always has the value of 1). Checks for disintegration are made yearly, but the main part of the algorithm is only triggered after lifetime reaches a certain number of years (given by the parameter $L_{fallen\text{-}tree}$). When this happens the fallen-tree ME is destroyed.

7.2.5 Felled trees

Felled tree MEs are identical to fallen-tree MEs except that they possess an extra attribute, that of sale-value. This value is the revenue obtained when timber from the stem is sold. Felled tree MEs are created in logging operations associated with the stand ME.

7.2.6 Stand

There is a single stand ME in each simulation which persists throughout the length of the simulation. It has ME constants x0 and y0 which give the stand origin and the ME-constants xlength and ylength which give the dimensions of the stand in x and y directions. It has a state-variable called lifetime and a rate-of-change variable called lifeyear.

The stand ME has an event called logging. Checks for logging are made annually, but the main algorithm is only used if the stand lifetime is equal to one of a specific set of values. The first logging occurs in a year set by the parameter t_{first} and the value of the interval is controlled by the parameter, t_{cycle}. (values for these parameters are given in Table 7.4). Two operations take place in each logging:

- individual trees are felled;
- felled trees are skidded to a point on the perimeter of the plot.

All trees that are above a critical stem diameter (set using the parameter d_{crit}) are selected for felling in a logging. Felling creates stand disturbance in the same way that natural disturbance events do. Skidtrail MEs are created for every tree that is felled. Each skidtrail axis originates at the base of the felled tree and terminates at an access point on the perimeter of the stand. The coordinates of the access point are set using the parameters x_{access} and y_{access}. The skidtrail MEs are rectangular in shape, the width of skidtrail created is determined by the parameter, W_{skid}. Trees below a certain diameter (given by the parameter d_{skid}) are assumed to be killed in the skidding operation. Values for logging parameters are given in Tables 7.4 and 7.5.

7.2.7 Skidtrails

Skidtrails have ME constants x0, y0, x1 and y1 that are used to store the co-ordinates of the skidtrail axis. They have the ME-constants vx0, vx1 ... vx3 and vy0, vy1,...vy3 to hold the co-ordinates of the vertices associated with the skidtrail's PDZ. They also have an event called overgrowth. Skidtrails are created in logging events associated with the stand ME and are destroyed by the overgrowth event associated with skidtrails. The overgrowth event results in the destruction of a skidtrail after a period of time given by the parameter $L_{skidtrail}$

Table 7.4: Parameters used in the model which have only one value. WS stands for Wanariset Sangai.

Parameter group	Parameter name	Source	Units	Value
Tree allometry	b - the ratio of crown radius to stem radius	Analysis WS data	m cm $^{-1}$	25
	f - tree volume form-factor	Analysis WS data	m^3 m^{-3}	0.45
	u - a coefficient giving the initial slope of the tree foliage relationship	Estimate	m^2	800
	v - the maximum value of foliage area that a tree can have	Estimate	-	400
Tree growth	r_{nbr} - the range within which trees are considered to be competing	Sensitivity analysis	m	15
Disturbance	N - the number of points within a gridsquare that are sampled to determine disturbance	Sensitivity analysis	cm cm^{-1}	4
	s_0 - the relative size of a tree in a potential damage zone below which no damage occurs.	Estimate	cm cm^{-1}	1.1
	s_1 - the relative size of a tree in a potential damage zone above which damage is certain	Estimate	cm cm^{-1}	3
	D_{crit} - the damage intensity in a gridsquare above which cohorts are reset	Estimate		0.75
Harvesting	x_{access} - the x-coordinate of the stand access-point	Arbitrary value	m	0
	y_{access} - the y-coordinate of the stand access-point	Arbitrary value	m	50
	t_{first} - the number of years after the start before the first logging takes place	Arbitrary value	years	5
	t_{cycle} - the number of years between subsequent loggings	Treatment	years	35,50 &70
	W_{skid} - the width of skidtrails created in a logging operation	Estimate	m	3
	d_{skid} - the critical size below which trees can be damaged in skidding operations	Estimate	cm	25
	$P_{skid\text{-}damage}$-the probability that a tree below dskid in a skidtrailwill be damaged			
ME lifetime	$L_{fallen\text{-}tree}$ - the number of years that a fallen-tree persists for after its creation	Estimate	years	4
	$L_{felled\text{-}tree}$ - the number of years that a fallen-tree persists for after its creation	Estimate	years	1
	$L_{skidtrail}$ - the number of years that a skidtrail persists for after its creation	Estimate	years	4

Table 7.5: Parameters that are disaggregated by species. These parameters have one value for every species group found in the model. Species groups are defined in Table 7.2.

Parameter group	Parameter	Source	Units	Value for each species-group					
				1	2	3	4	5	6
Tree allometry	***a*** - the ratio of crown-point to total height	Analysis of data from Wanariset Sangai	$m\ m^{-1}$	0.55	0.55	0.55	0.55	0.55	0.55
	o - the maximum height that a tree can attain	Literature	m	60	70	70	85	70	50
	e - the initial slope of the curve relating tree height to diameter	Estimate	$m\ cm^{-1}$	200	200	200	200	200	400
Cohort	***lo*** - the lai below which there is no growth of the cohort	Estimate	$m^2\ m^{-2}$	2.5	2.5	2.5	2.0	2.5	0.0
	l_1 - the lai above which there is no growth of the cohort	Estimate	$m^2\ m^{-}$	8.0	6.0	6.0	6.0	6.0	2.0
	$T_{maturity}$ - the number of years a cohort must grow for before trees reach 10 cm stem diameter	Estimate	years	21	16	16	16	14	9
	E - the number of trees that a cohort will produce on attaining maturity	Estimate	-	0.4	0.4	0.4	0.4	0.4	0.6
Harvesting	d_{crit} - the critical stem diameter for logging	Literature	cm	70	70	70	70	70	NA

Table 7.6: Parameters that are disaggregated by species and by size class. These parameters have one value for every combination of species-group and size class. Species groups are defined in Table 7.2.

Parameter group	Parameter	Source	Species-group						
			Size class (cm)	1	2	3	4	5	6
Tree growth	q - coefficient used in Equation 7.8	Analysis of Berau data	10-20	-0.35	-0.15	-0.15	-0.16	-0.20	-0.40
			20-30	-0.35	-0.15	-0.15	-0.16	-0.20	-0.40
			30-50	-0.35	-0.15	-0.25	-0.16	-0.20	-0.40
			50-70	-0.40	-0.15	-0.25	-0.16	-0.20	-0.40
			>70	-0.40	-0.15	-0.25	-0.16	-0.20	-0.40
	k - coefficient used in Equation 7.8	Analysis of Berau data	10-20	0.75	0.80	0.80	1.00	0.50	1.40
			20-30	0.75	0.80	0.80	1.00	0.50	1.40
			30-50	0.85	0.60	0.95	0.85	0.30	0.10
			50-70	0.85	0.50	0.70	0.70	0.30	0.10
			>70	0.75	0.50	0.00	0.00	0.20	0.00
Natural disturbance	P_{im_t} - the probability that an individual will initiate natural disturbance	Estimate	10-20	0.028	0.028	0.028	0.028	0.028	0.04
			20-30	0.032	0.032	0.032	0.032	0.032	0.04
			30-50	0.020	0.020	0.020	0.020	0.020	0.04
			50-70	0.020	0.020	0.020	0.020	0.020	0.04
			>70	0.020	0.020	0.020	0.020	0.020	0.04

7.3 FORMAL MODEL DESIGN

An excerpt from the formal SYMFOR design for the model described in this chapter is given in Figure 7.3. The excerpt specifies the representation of individual trees used in the model. Primitives used in model designs are described in Table 5.1. It can be seen that there is a one to one correspondence between concepts described in Section 7.2.1 and statements in the file.

Figure 7.3: Excerpt from the formal SYMFOR Model Design for implementing the model described in this section.

```
entity( tree).
pattern(tree, irregular).
statevar(tree, dbh).
me-constant(tree, x).
me-constant(tree, y).
conclassifier(tree,species).
dynclassifier(tree, sizeclass).
intermediate(tree, height).
intermediate(tree,crownpoint).
intermediate(tree, basalarea).
intermediate(tree, shadeindex).
intermediate(tree, dbhincr).
event(tree, treefall).
delta(tree, dbh, [ [+,dbhincr] ]).
modulechoice(tree, sizeclass, sizeclass1).
modulechoice(tree, height, height2).
modulechoice(tree, crownpoint, crownpoint1).
moudlechoice(tree, basalarea, basalarea1).
modulechoice(tree, volume, volume2).
modulechoice(tree, shadeindex, shadeindex2).
modulechoice(tree, dbhcincr, dbhincr2).
modulechoice(tree, treefall, treefall2).
```

7.4 PROCEDURE

The investigation was carried out using SYMFOR release 2.21. The software was mounted on a Dell Latitude Xpi laptop. The computer used had a Pentium processor with a clock speed of 100 MHz and was equipped with 16MB of RAM.

Ten replicate runs were performed for each combination of plot and treatment. The length of each simulation was 160 years giving 4, 3 and 2 cutting cycles for the three treatment cycle lengths used (35, 50 and 70 years). In each run the volume extracted in each successive logging was noted. The statistic of Mean Annual Harvest (MAH) was calculated for each plot-treatment combination using the formula given in Equation 7.11.

$$MAH = \frac{\sum_{i=2}^{i=n} V_i}{n \cdot c} \qquad \text{Equation 7.11}$$

where MAH is the Mean Annual Harvest in $m^3\ ha^{-1}\ yr^{-1}$, i is the harvest number, n is the number of harvests in 160 years, V_i is the timber volume extracted in harvest i in $m^3\ ha^{-1}$ and c is the cycle length in years. This is a similar statistic to Mean Annual Increment except that it uses harvested volume (rather than standing volume) and the first harvest is excluded from this analysis, as it is not representative of the management of logged-over forest. Mean and standard error values for increment were calculated for each treatment/plot combination. An analysis of variance was performed on the increment data using the MINTAB statistical package. The ANOVA model used was that described as 'Complete randomised block design' by Sokal and Rolf (1995). More detailed results from a single typical simulation were collected and plotted.

7.5 SITE DESCRIPTION

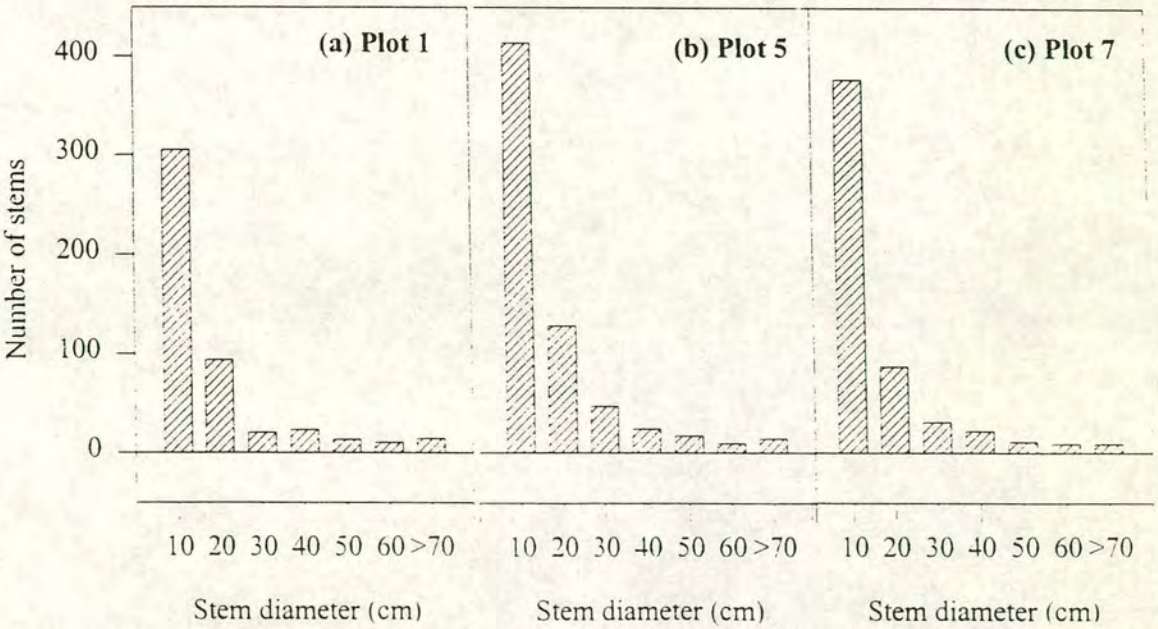
Stand Initialisation data came from the Wanariset Sangai Research Forest of Central Kalimantan in Borneo (1°29'S, 112°31'E). Data from 3 of the 15 experimental plots established at Wanariset Sangai were used (Plots 1, 5 and 7). Each plot is 100m x100m. The data used is taken from plots before any logging treatment was applied.

This part of Kalimantan receives annual rainfall of 3000 mm and average daily temperature of 25 °C (Clearwater, 1996). Soils are Ultisates (US Department of Agriculture classification) or Acrisols (FAO classification) and are typical of forest of this kind in Borneo (Proctor, 1994). Plots 1 and 7 are in valleys and have a stream running through them. Plot 5 is on a steep slope of 20-40°.

The vegetation on the plots is primary forest (*i.e.* it is thought that little or no previous logging has taken place). The species richness is of the high, with a total of 1322 different taxa reported in 13 plots in which collections were made (Argent *et al.*, 1993). The most important family is the *Dipterocarpaceae*, which constitutes 14% of the trees present.

The initial size class distributions of the three plots used in the simulation are shown in Figure 7.4. It can be seen that Plot 5 is more highly stocked in the lower size classes.

Figure 7.4: Size class distributions for trees in 3 plots of Wanariset Sangai research forest.



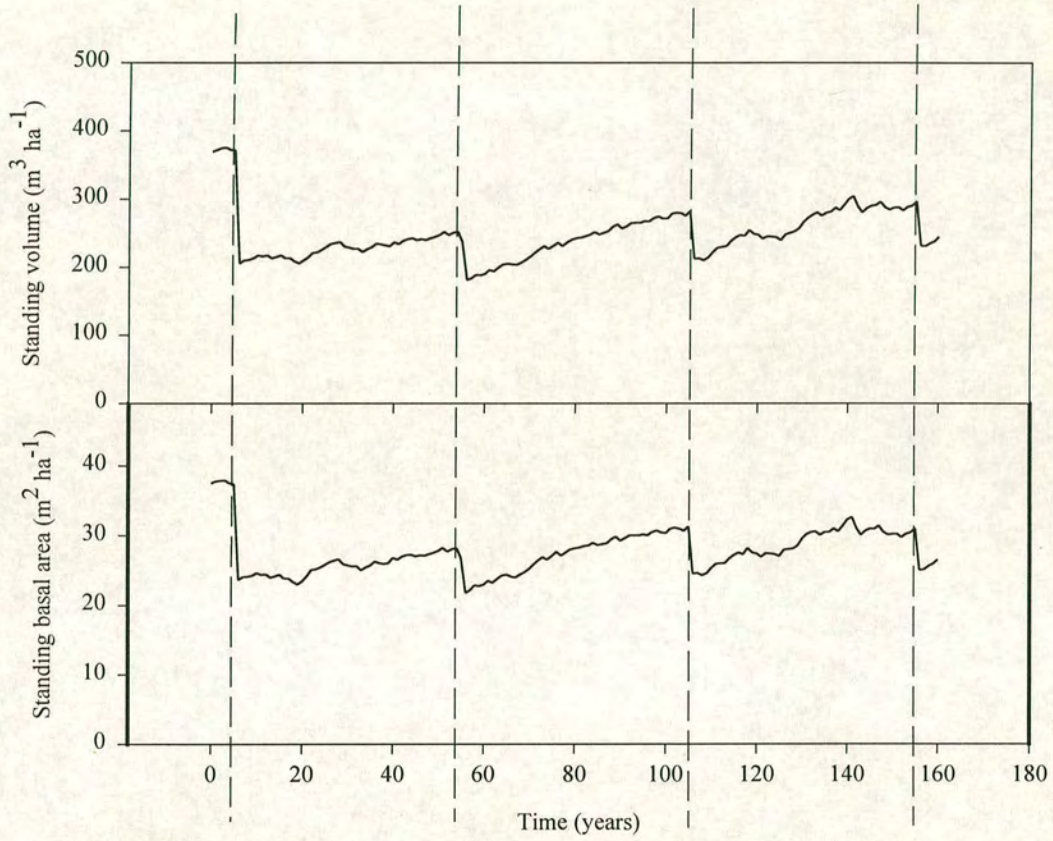
7.6 RESULTS

The graphs contained in Figures 7.5 to 7.7 show how various stand characteristics change in a typical run of the model. The run selected used a Stand Initialisation file for Plot 5 and had the felling cycle set for 70 years. Loggings occur in years 5, 55, 110 and 145 of the simulation.

Figure 7.5 shows production characteristics of the stand. Standing volume falls from 370 m^3 to 220 m^3 on the first logging. After this volume increases with smaller decreases associated with subsequent loggings. Standing basal area shows a very similar pattern to standing volume.

The number of standing trees in different size classes through time is shown in the graphs of Figure 7.6. Figure 7.6 (a) uses a linear scale while Figure 7.6 (b) uses a log scale. The log scale is better for illustrating changes in the larger size classes. The size classes used were given the following names: poles (10-30 cm dbh), sub-canopy (30-50 cm dbh), main canopy (50-70 cm dbh) and emergents (>70 cm dbh). The graphs shows that most of the year-to-year variability in the total number of standing trees is due to variability in the number of poles (the other size classes change much more slowly). About 140 poles (21 % of the total number of trees) are lost in the first logging. Numbers then decline for a period of around 20 years, after which time there is a sudden increase in number to around 500. Similar patterns are associated with subsequent loggings.

Figure 7.5: Stand production characteristics obtained from a typical run. Harvesting operations take place in year 5, 75, and 150 and are indicated with dashed lines.



Further stand characteristics are shown in Figure 7.7. Disturbance intensity varies substantially from year to year. In most years it is between 0 and 15%. In the first logging years it is above 40%, but subsequent loggings fall within the normal range of disturbance. Recruitment also varies a lot between different years. There is a pronounce peak 15 years after the first logging, but no pattern is discernible for subsequent loggings. Mortality shows a very similar pattern to disturbance intensity. In most years it is between 0 and 20 trees with considerable year-to-year variability. In logging years it is much higher at 80 or more trees.

Figure 7.8 shows the variability between replicate runs in Standing Volume through time. In each case the same Stand Initialisation file (Plot 5) and the same set of parameter values (felling cycle = 70 years) were used. The Figure shows considerable variation among replicates.

Figure 7.6: Dynamics of different size-classes of tree obtained from a typical simulation run. Harvesting years are indicated with dashed lines. Four size classes are shown: poles (10-30 cm dbh), sub-canopy (30-50 cm dbh), main canopy (50-70 cm dbh), and emergents (>70 cm dbh).

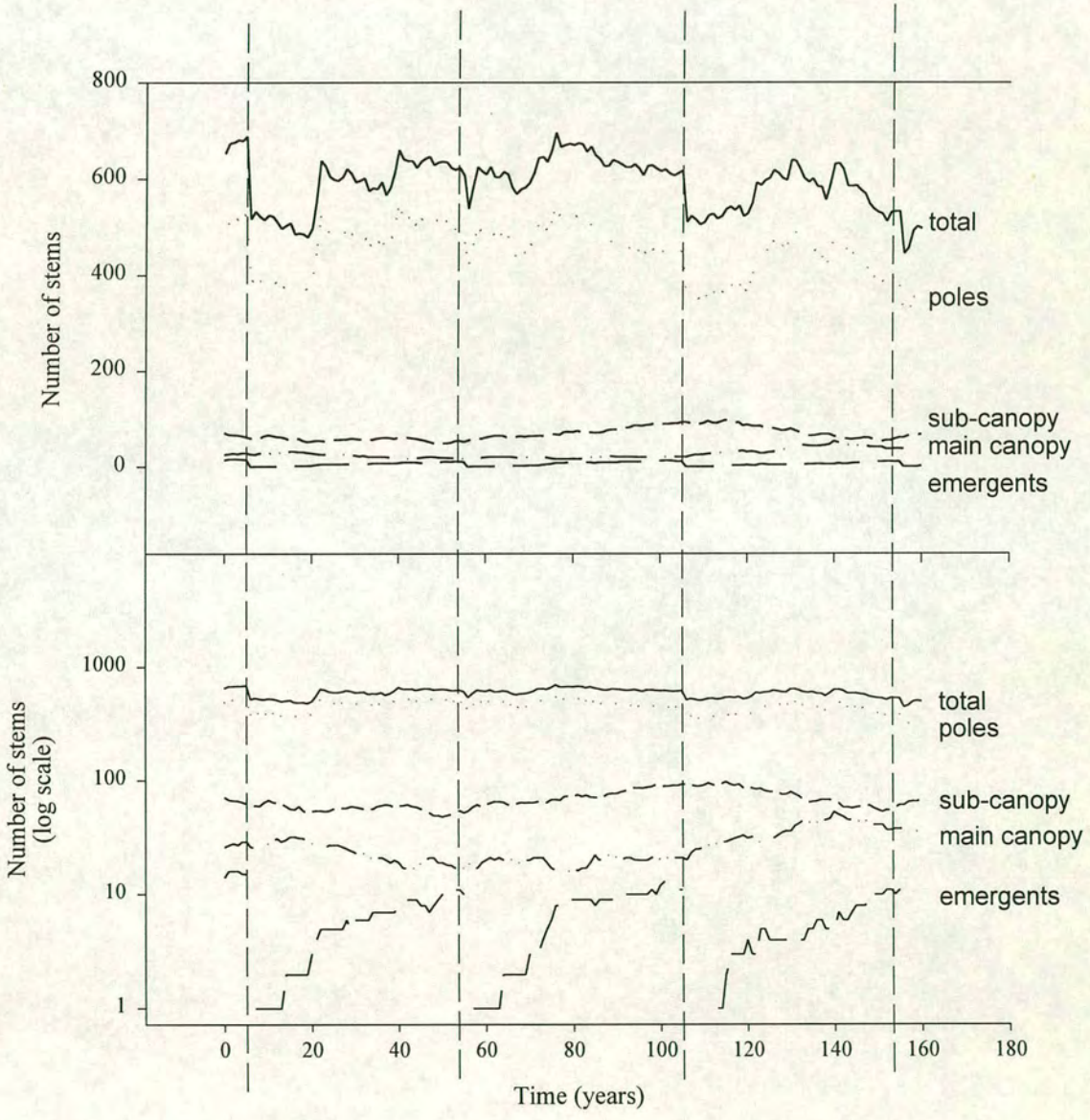


Figure 7.7: Further stand characteristics obtained from a typical simulation run. Harvesting years are indicated with dashed lines.

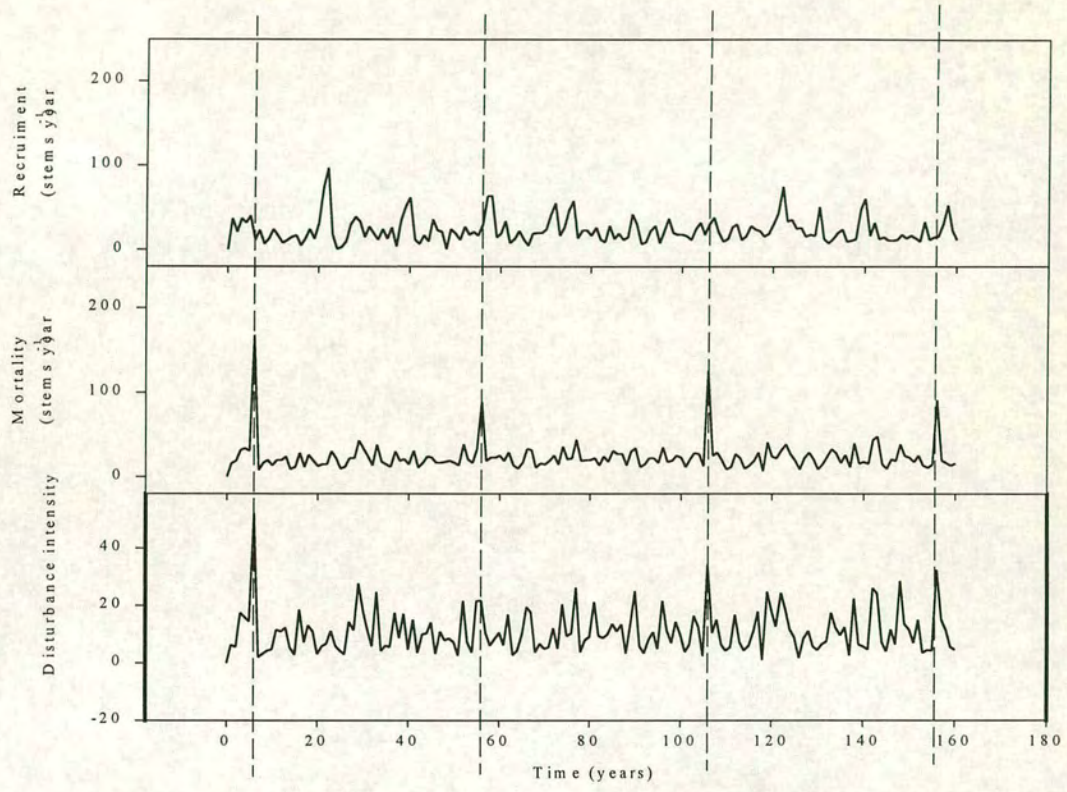
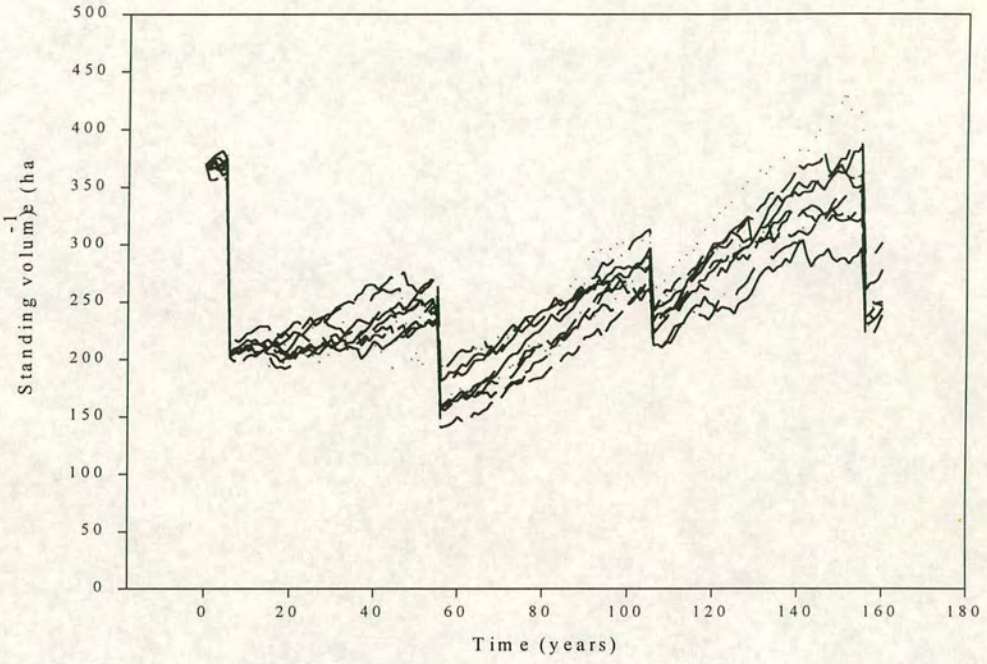


Figure 7.8: Graph showing the variability in Standing Volume obtained between replicate runs that used the same SID and Parameter Set.



The graphs shown in Figures 7.4 are frequency distributions giving the number of stems in different size classes above 10 cm dbh. Plot 5 has higher stocking in all size classes.

Tables 7.7 and 7.8 give details of the MAH rates obtained in the experiment. Table 7.7 gives mean values of the statistic obtained for different plot-treatment combinations while Table 7.8 summarises the results of the analysis of variance performed for MAH rates. The latter shows that both plot and treatment are highly significant ($p<0.000$).

Table 7.7: MAH rates for different Plot/Felling cycle combinations. Each cell contains the mean and the standard error. In all cases n = 10.

	35 year felling cycle	50 year felling cycle	70 year felling cycle	Total
Plot 1	2.29 ± 0.111	2.31 ± 0.096	1.96 ± 0.106	2.19 ± 0.107
Plot 5	1.39 ± 0.050	1.43 ± 0.050	1.08 ± 0.074	1.30 ± 0.074
Plot 7	1.31 ± 0.066	1.14 ± 0.045	1.00 ± 0.056	1.15 ± 0.068
Total	1.67 ± 0.163	1.62 ± 0.173	1.35 ± 0.155	1.55 ± 0.168

Table 7.8 summarises the results of the analysis of variance performed for MAH rates. Both Plot and treatment are highly significant ($p<0.000$).

Table 7.8: Analysis of Variance for MAH rates obtained in the simulation experiment

Source	DF	SS	MS	F	p
Plot	2	18.9878	0.44154	182.27	0.000
Treatment	2	1.8313	0.9157	17.58	0.000
Error	85	4.4273	0.0521		
Total	89	25.2445			

7.7 DISCUSSION

7.7.1 Results from individual run

Figure 7.6 contains graphs showing the number of trees in different size classes through time. It shows that logging has most effect on the poles in the stand, and relatively little effect on the higher size classes (other than the emergents, which are removed as part of the logging). At the time of logging a larger proportion of the area in a forest stand will be cleared. This is consistent with Figure 7.7, which shows the disturbance intensity and mortality within the stand. The two often peak in logging years.

One consequence of the loggings is that many saplings below 10cm dbh are destroyed. This can result in a temporary reduction in recruitment to the individual-based submodel. This behaviour is reflected in the graphs in Figure 7.6, where the drop in poles after a logging is followed by a flat or declining period of approximately 20 years. Some time after logging seedlings begin to establish in the cleared area. This may happen at roughly the same time throughout the area. When these seedlings grow to become trees of 10 cm dbh (and so become recruits to the individual based model) there is a surge in recruitment.

7.7.1.1 Volume production

Predicted first cycle yield in all replicates is of the order of 150 m^3 ha^{-1} while yields from the second cycle are lower (of the order of 40 m^3 ha-1). The predicted volumes extracted from the first cycle loggings are high compared with figures in the literature for tropical forests (*e.g.* Bertault and Sist, 1995; Silva *et al.*, 1995). This discrepancy probably arises for three main reasons.

First, it seems likely that Wanariset Sangai is unusually heavily stocked with large trees. The large first cycle yield is consistent with the size distribution of the stand at the start of each run - a relatively high number of trees are greater than 70 cm dbh. Each run was initialised with data from PSP's established in Wanariset Sangai Research forest. It seems likely that this area of forest is particularly high in volume. The large number of big trees also helps explain the large reduction in standing volume. Some of the loss is due directly to the removal of the big trees while some of the loss occurs through damage and subsequent death trees in the residual stand caused by felling and extraction operations. As the number of trees removed is comparatively large (~ 25 trees) there are a large number of skidtrails and collectively these cover a greater proportion of the ground surface of the stand and this leads to greater damage. In subsequent loggings many fewer trees are removed. This limits damage to residual stand and results in a much lower total volume being extracted. The drop in standing volume is consequently much lower.

Second, in the simulations strict adherence to the diameter limits for felling is assumed. In the field there is evidence that the prescribed regulations may not always be followed precisely. Alder (1995) distinguishes between *normative* and *empirical* simulation of harvesting, the former being the logging that would take place if regulations were followed to the letter and the latter being the logging practices observed in the field. This will be a problem with many experimental PSPs as commercial and political pressures may mean that applied treatments may not reflect reality in the field.

The third reason that the volumes extracted may be large compared with those obtained in practice may relate to details concerning the form and quality of stems that are not captured in the model. If a stem is bent, diseased or imperfect in some other way then a decision may be made not to fell it. The model holds no information of this kind for trees. One way in which the model could be improved would be to find the proportion of stems that are imperfect from field measurements and to use this proportion in the model to select stems for felling.

7.7.2 Variability among replicate runs

The most striking thing about Figure 7.8 is the amount of variability shown among replicate runs that used the same Stand Initialisation file and the same set of parameter values. This occurs because of stochastic elements within the model. For example, natural disturbance initiation and resulting mortality is handled stochastically.

Other models, such as size class models, may produce behaviour that varies less or not at all between replicate runs. However, they may achieve this by suppressing behaviour associated with the largest trees in the stand. This is because, as is discussed in Section 2.2, aggregation of individuals will usually produce bias towards median individuals and away from more extreme individuals. There are likely to be a few large trees in a stand so that often the size-class used to contain them will be wide. Moreover, the behaviour of the largest trees is disproportionately important - they contain a relatively large proportion of the standing volume of the stand (which will be immediately lost when they fall) and they are more likely to dislodge other trees when they fall (Clark and Clark, 1996). The result of aggregation may be that size-class models systematically underestimate factors such as stand disturbance and the variability that would occur in a real stand. These errors are examples of aggregation errors (Huston, 1994; O'Neill and Rust, 1979).

Another important point when discussing stochasticity is that field based trials of silvicultural systems may be prone to the same problem, in that the results from a particular PSP may be sensitive to rare and irregularly timed events. They compare unfavourably with modelling approaches in that replication to assess the significance of these events involves establishing new plots. In both modelling and purely empirical approaches variability can be reduced by increasing plot size to greater than 1 ha *e.g.* to 4 ha. In the case of an individual-based model (IBM) this also has the advantage of lessening edge effect. The alternative approach of using wrap-around techniques, has the difficulty that it will tend to bias results by exaggerating the influence of phenomena such as treefalls taking place at the edges (Alder, 1995).

7.7.3 Treatment and plot effects on MAH rate

Table 7.8 shows the results of an analysis of variance on MAH obtained for different plots and for different treatments (lengths of felling cycle). The treatment factor is not significant. The Mean Square for the plot factor is much higher than that obtained for the treatment factor. This suggests that Plot is by far the more important factor (it is inappropriate to test the significance of F for the Plot effect using this model of ANOVA (Sokal and Rohlf, 1995)).

While it is intuitive that the volume extracted in the first logging will be very strongly related to initial stocking, this is not the case for subsequent loggings. This is because it is possible that the trees in the plot may respond to logging and canopy opening by increasing their growth rates. The increased growth may in some measure lessen the impact of different initial stocking density on subsequent yields. The results from this simulation experiment suggest that any increased growth is insufficient to compensate for different initial stocking.

7.7.4 Comparison with the design of other individual-based models

The model design used in this investigation differs from that of other IBMs in two main ways: regeneration is captured using quite a sophisticated submodel and an attempt is made to capture the spatially and temporally clustered nature of mortality in forest stands. IBMs of the JABOWA / FORET type (often referred to as gap models) use simpler methods for capturing regeneration. In these methods a list of eligible species for establishment in each gap is determined based on gap characteristics such as presence of leaf litter, abundance of seed predators and the characteristics of the different species in the model (Shugart1984). There is no delay between the occurrence of appropriate conditions and the creation of new individuals. This is perhaps plausible if the threshhold for treatment as individuals is low (*e.g.* of the order of 2 cm dbh) but less so if it is high (of the order of 10 cm dbh). The more complex solution advocated by Shugart (1984) (use of size-classes) is problematical because of the nature of the seedling/sapling population in a tropical forest. Because of irregular seed production and seedling growth it can often happen that particular size-classes are not represented in a gap, and size-class models, as well as being computationally inefficient can produce pathological behaviour when empty size classes occur(as discussed in Section 2.2).

In most gap models each individual tree has a probability of undergoing mortality that is unaffected by the occurrence of other tree mortality in its vicinity. In a real forest up 75 % of mortality can be secondary in nature (Van Der Meer and Bongers, 1996) *i.e.* caused by the impact of other falling trees within the stand. Capturing this is important as it affects the distribution of disturbance intensities caused by mortality and treefall within the stand. High local disturbance intensities will be more common when clustering is captured and this is the situation likely in a real forest. As discussed in Section 2.1 local conditions are very important in determining factors such as regeneration and nutrient cycling so that failing to capture clustering of mortality can be seen as a serious omission.

7.7.5 Possible improvements to model design

This Section considers possible changes to the model which could result in improved predictions of growth and yield. As discussed in Section 2.3.2, increases in complexity may not *necessarily* result in a better model. However, the things discussed may *potentially* be of use in a model. Much of the material in this section draws on Section 2.1. However, where relevant material is repeated in the interests of readability.

The model could be changed to capture more detail of tree allometry. In the model trees belonging to the same species group are assumed to show the same morphological development. This arises because only one state-variable (stem diameter) is used to capture the developmental stage of the tree. As a consequence all the dimensions of an individual tree are determined once dbh is known. This means that trees belonging to the same species group that have the same diameter will always have the same height. One problem with this is that in the field it is commonly observed that trees from some species groups will have different growth forms depending upon the environment in which they have grown. As is discussed in Section 2.1.2, open grown trees may have sympodial development while trees growing in a forest with quite a high degree of canopy closure may show monopodial development (King, 1995).

Uptake of resources other than light could be considered in the model. Water and nutrients such as nitrogen, phosphorous, calcium or magnesium may also be important determinants of tree growth. Given the critical role of soil organic matter in nutrient cycling (Whitmore, 1989) of tropical soils it may be useful to model litter production and organic matter decomposition processes. Mycorrhizal dynamics could also be included. It has been shown that mycorrhiza growing in association with tree roots can be very important in determining uptake of phosphorous (Alexander *et al.*,1992). Moreover mycorrhizal populations can be severely reduced by the environmental changes associated with logging and can take time to recover (Alexander *et al.* 1992; Ahmad, 1996).

The competition index used could be changed to take into account know characteristics of light transmission and absorption within the forest canopy. There may be problems in modelling competition for light with the chosen competition index (the distance weighted size ratio). The formulation of this index implies that a tree's ability to consume resources (and so diminish resources available for consumption by other trees) is related to its stem diameter. In reality very large trees may have crowns that are so far away from the smaller trees that they may intercept a small fraction of the light that would otherwise be received by the smaller trees. One way in which the competition index could be improved might be to use stem sapwood area instead of stem diameter. Sapwood area may more closely reflect the dimensions of the crown and so reflect consumption of light and other resources better.

The model could be improved by modelling growth in a stochastic manner. Currently the model uses a deterministic approach, despite the fact that there was a large amount of unexplained variation when the growth functions were fitted (such variation is often found with permanent sample plot data (Sianturi, 1996; Alder 1995)). The result of using the deterministic function is that there may be bias towards median growth rates. This may cause problems. Volume increment is a non-linear function of diameter increment. The main problem in using a stochastic function is capturing the correlation between successive diameter increments for individual trees that is observed in the field. This feature means that an element that is used to modify the values of diameter increment must be remembered between time-steps (*i.e.* it must be a state-variable). This raises the problem of determining values for the element to be used in model initialisation.

In the model regeneration is not constrained by site conditions or by supply of propagules to the soil surface. Loss in soils ability to support seedling growth after logging can occur due to soil-compaction, loss in organic matter through increased rates of decomposition, increase soil temperatures and increased likelihood of soil drying out (Nussbaum *et al.*, 1995). It is intuitive that propagule supply is important and indeed other IBM's have sought to capture propagule production and dispersal by individual trees (*e.g.* Luan, 1994). However, the situation is more complex in a species-rich forest and in a study of the importance of site conditions and propagule supply in Sabah, Pinard *et al.*(1996) concluded that the latter was much less important than the former.

The model could be changed to take account of plot topography in any of its calculations. Topography may be important in a number of ways. First there is evidence that patterns of tree growth and development may differ between ridges and depressions (*e.g.* Ashton, 1964; Basnet, 1992). This may be related to increased interception of radiant energy or to better drainage (and hence aeration) of the soil on ridges. Second, topographical information is useful in predicting erosion. Loss of topsoil following logging can be substantial (Baharuddin *et al.*, 1995) and can result in large changes in soil properties (Nussbaum *et al..*, 1995). Third, decisions on the routing of skidtrails, and even on which trees to extract in a real forest may be taken using topographical information (Pinard and Putz,. 1996).

8. Discussion

This section considers the extent to which SYMFOR can be said to meet the requirements articulated in Chapter 3. Evaluation of the success of SYMFOR in meeting these requirements is useful because it indicates the *design quality* (Budgen, 1994) and also because it provides a useful way of structuring a comprehensive review and discussion of preceding material.

8.1 FOREST PRODUCTION ESTIMATES

Long-term estimates of forest production (specified in Requirement 1) are achieved by SYMFOR by use of simulation models to project modelled stands into the future. Forest simulation is now widely recognised as an important tool for assessing future forest production (*e.g.* Davis and Johnson, 1987; Adlard, 1989; Vanclay, 1994, 1995; Alder, 1995, Kimmins, 1997). A simulation approach is superior to a purely empirical approach because of the problems of obtaining harvests representative of second and later cycle fellings. There are basically two problems. First, large scale logging has commenced relatively recently in many tropical countries. This means that it can be difficult to find plots of forest which have been established long enough to have experienced second and later cycle harvests (Palmer and Synnot, 1992) . Second, management techniques have changed since the first plots were established. For example timber extraction now often employs heavy duty machinery that was previously unavailable (Whitmore, 1988). This means that previously established plots are unrepresentative of current silvicultural practices.

Requirement 2 specifies that the framework should be able to cope with spatial variation in forests. The framework supports this in three ways: by allowing the redesign of models; by allowing models in the framework to be recalibrated; and by allowing model runs to be initialised with data from different plots. Redesign of models is appropriate when there are *qualitative* differences in the functioning of forest areas (*i.e.* differences in the set of processes that are important within the stand). For example, in one area of forest the primary limitation on growth of trees may be seasonal dryness, while in another there limitation on growth may be caused by the rate of organic matter turnover. Different models with different relationships and parameters are required to deal with each of these forest types. Recalibration is appropriate when differences that occur between forests are *quantitative* in nature (*i.e.* differences in the rate at which processes in the stand occur for a given set of conditions). For example, turnover of organic matter may occur more slowly in one stand than another at a given temperature because the stand drains poorly and the soil is waterlogged for much of the time. Initialising the model with different stand states is an appropriate way of dealing with variation in stand structure arising for historical reasons (as opposed to differences in forest functioning). For example, some areas may have undergone major disturbance relatively recently while others may not have experienced major disturbance for a long time. This may affect both stand structure and species composition (Cannon *et al.*, 1994).

Requirement 3 stipulates that the framework should be able to cater for variation in harvesting intensities and harvesting practices, while Requirement 4 specifies that it should be able to predict the impact of silvicultural operations other than harvesting. These requirements are met by using a representation of the stand in which the stand structure is explicitly represented and in which the locations of trees are represented. The representation of stand structure means that it is possible to calculate the number of trees that will be harvested (and hence the volume of harvested timber) under different harvesting intensities more accurately than if an aggregate stand representation were used. The representation of tree locations means that the impact of different harvesting practices which seek to minimise damage to the stand created in harvesting operations can be assessed more accurately than when an aggregate representation is used. These methods are becoming increasingly important (Pinard and Putz, 1996).

8.2 REPRESENTATION OF FOREST PROCESSES

The model described in Chapter 7 exemplifies many of the approaches described in this section that can be used to capture forest processes. However, SYMFOR is designed to permit a lot of flexibility in model structure to allow the framework to cope with a wide range of circumstances. This means that details of how processes are represented, and indeed, which processes are represented, may vary from model to model. There is therefore a need to describe the general approaches that can be used.

The way that processes are represented in a model depends on two things: the way that trees are represented and the nature of the processes themselves. SYMFOR models use two kinds of modelled entity to represent the trees of the stand: 'cohort' modelled entities (which represent populations of trees that establish at the same point in time) and 'tree' modelled entities (which represent individual trees). A cohort representation is used for the earlier stages of growth and development such as the seedling and sapling stage. Seedlings and saplings that establish at the same point in time are usually similar in size, so that an aggregated representation can be used without substantial loss. An individual-based representation is used for the later stages of development (*e.g.* trees above 10 cm stem diameter), when size homogeneity breaks down. The spatial disaggregation necessary for capturing the localised nature of many stand processes is introduced by breaking the stand into a contiguous set of gridsquares. Each gridsquare represents a small portion of the stand *e.g.* a 10 x 10 m square area. Each tree and cohort that is represented is associated with one of these gridsquares. Species disaggregation is achieved for cohorts by using distinct units to represent cohorts of different species (Requirement 7). Gridsquares themselves are modelled entities capable of possessing attributes.

In accordance with Requirement 5, SYMFOR models capture disturbance in forest stands. As discussed above, SYMFOR models are spatially disaggregated so that a single representation unit covers only a small part of the stand. Over small spatial scales disturbance is an essentially intermittent phenomena (*i.e.* treefalls occur irregularly). For this reason disturbance is captured using the concept of 'event' described in Section 4.2. Natural disturbance events are associated with tree modelled entities, while logging disturbance event is associated with the stand modelled entity (because of the higher level of organisation involved in stand logging). The algorithm used in the evaluation of each event has two parts, one that determines *disturbance initiation* and one that determines *disturbance impact.* Event initiation is stochastic (*e.g.* natural treefalls) or deterministic (*e.g.* stand logging). In the case of natural tree falls every tree is assigned a probability of initiating a disturbance event every year. A Monte Carlo technique is then employed to determine if an individual actually initiates an event in a year This is a standard technique for gap models (Shugart, 1984). In the case of a logging the initiation is determined by the felling cycle, and this is a parameter set by the user.

Disturbance impact is captured by changes to the stand state. Such changes include destruction of tree representation units (*i.e.* individual trees or cohorts) and change in the value of model state variables (*e.g.* soil organic matter). An individual tree fall creates a fallen tree modelled entity possessing a *potential damage zone*. Tree representation units (both individual trees and cohorts) within this potential damage zone are subject to secondary mortality. Each tree has a probability of undergoing mortality based on its size compared to that of the tree initiating the disturbance. This approach means that models are capable of capturing the localised nature of disturbance (Requirement 8). In the past individual-based models have neglected secondary mortality of this kind, but this is now changing (Kurpick *et al.*, 1997; Kohler and Huth, 1998).

Requirement 6 specifies that SYMFOR models should capture recovery from disturbance. In reality this is composed of a number of stages, each or all of which can be explicitly captured in a SYMFOR model. The stages include:

- seed production and dispersal - this can be handled either as a series of events (when it is intermittent rather than continuous in nature) or as a set of rates of change (when production occurs continuously in the stand). Seed production can be a stand or gridsquare attribute or an individual-tree attribute. The former two are appropriate when seed distributions are related to site conditions rather than propagule sources. This is effectively the approach taken in many of the earlier gap models (Shugart, 1984). The latter is appropriate when the seed distribution resulting from dispersal is not homogeneous but is centred on individual trees. In this case seed dispersal must also be modelled. This is the approach taken in Levin *et al.*(1984) and Luan (1994).

- seedling and sapling development. As mentioned above, a cohort representation is used for these phases of tree development. This aggregation is appropriate because early on in development individuals of the same cohort are similar with respect to size and other characteristics such as nutrient uptake, so that it is inefficient and unnecessary to represent them individually. Development is handled using two cohort state-variables, one reflecting the size or developmental stage of members of the cohort and another giving the number of individuals represented by the cohort. Changes to the values of the state-variables can be calculated in various ways (Alder, 1995).

- development of asymmetry within cohort populations. This takes place in the stages of development for which an individual-based representation is used. It can be captured by modelling competition between trees. In one scheme for achieving this each tree modelled entity possesses three attributes: shadeindex (a competition index); dbhincr (stem diameter increment); and dbh (stem diameter). Shadeindex is an intermediate variable. Its value reflects the amount of competition that the tree is facing from other trees in the stand. It can be derived in various ways, some of which may explicitly involve modelling light transmission through the forest canopy (see Dale, Doyle and Shugart, 1985). Dbhincr is a rate of change. It is dependent upon the value of dbh and the value of shadeindex. It is used to increment the value of the state variable dbh, which represents the stem diameter of the tree.

Any or all of the stages of recovery from disturbance can be explicitly represented in a model. However, usually only one or a small number of the stages are important. Even when the impact of the stage/process is substantial, it is only if there is a systematic effect that is modellable that the process should be explicitly represented. For this reason seed or seedling predation may often be omitted.

8.3 DATA COLLECTION

Models in the framework use standard permanent sample plot (PSP) data as far as possible. This is stipulated by Requirement 6. This can be achieved within SYMFOR by choosing to represent processes that can be calibrated with PSP data in models, and avoiding the representation of processes that cannot be calibrated with such data. For example, diameter increment of trees can be calibrated with PSP data, while canopy photosynthesis cannot be. (Note that this is a constraint imposed by the framework user and not the framework itself - it is perfectly possible design a model that represents physiological processes using SYMFOR mechanisms for model design). The relationship between forest models and data is further discussed in Vanclay (1994) and Alder (1995).

One problem with this approach of omitting physiological processes is that insight in this area cannot be used to make model predictions. This is potentially a disadvantage because factors associated with global climate change (*e.g.* rise in temperature) may mean that relationships which currently hold good (*e.g.* between competition indices and stem diameter increment) may not hold good in the future. However, one of the main purposes of SYMFOR is to facilitate comparison of alternatives silvicultural systems, and often the rankings determined in an analysis will be robust. For example, in a comparison of reduced impact logging techniques and conventional logging technique the winner will probably be unaffected by climate change.

Where some extension to current data collection procedures is necessary, care is taken to ensure that extra data collection is minimised and that resources required for collection are similar to those currently in use. For example, disturbance resulting from tree falls is captured in SYMFOR models. Data collection procedures developed for the purpose of calibrating the representation of disturbance in the model is described by Clearwater (1996). They involve mapping the damage created by individual tree falls. However, the main resource used in the procurement of this extra data is labour, and no specialised or expensive equipment is required.

Requirement 7 specifies that data requirements for calibration should not be excessive. One way in which this can be achieved is by use of 'targeted' models *i.e.* models designed for a particular forest type and designed to provide answers to a particular question. This is the approach favoured by Kimmins (1996). Targeting means that the number of processes represented in a model can be reduced to those that are strictly relevant. For example, the management operations that need to be parameterised can be restricted to those under investigation. Similarly parameters describing the relationship between water uptake and growth can be omitted in areas where there is no water limitation on growth.

Requirement 8, that recalibration of models should be infrequent can be achieved by using 'general' model designs. Running and Coughlan (1988) and Bossel *et al.* (1989) describe models designed to have a high degree of generality by virtue of the fact that they represent all the processes that the authors deem *generally* important (rather than all those relevant for a particular site or a particular purpose). An increase in detail may allow a model to simulate a wider range of behaviour. For example, in some model individual tree growth may be related to tree diameter and a competition index of some kind. In another individual tree growth may be related to tree diameter, competition index, and water uptake. The first model is suitable for areas where there is no water limitation on tree growth. The second is appropriate for areas with water limitation AND areas without water limitation. However, one disadvantage of using the second model in areas with no water limitation is that some of the parameters needed are effectively redundant (in that they do not lead to better predictions), and it may be expensive to obtain values for these parameters.

The decision on whether Requirement 7 or Requirement 8 is more important in the design of an individual model is essentially an economic one. The total effort required for parameterisation is a function of the frequency with which models need to be recalibrated and of the effort that must be expended on each recalibration. In some situations a more detailed forest representation will not need to be recalibrated any less frequently than a more complex one. This may occur if the extra detail is insufficient to capture variation that occurs across the range of circumstances that predictions are required for. For example, a model which captures water relations and that is calibrated for a particular site may be unable to make good predictions for another site. This can happen because other factors (*e.g.* temperature, soil conditions or altitude) are omitted. It may be more cost effective in such cases to simply recalibrate the model rather than increase the detail of the model representation (and so increase the parameter requirements).

8.4 FRAMEWORK ARCHITECTURE

The requirement that SYMFOR support collaborative modelling (Requirement 9), is met by the satisfaction of a number of more specific requirements from the list given in Chapter 3. Two of these affect the ease with which model designs can be changed to take account of new circumstances: Requirement 10 specifies that the framework provide support for creation of new model designs and Requirement 11 specifies that the framework should provide support for sharing and reuse of model content. As discussed in Section 2.3.2 changes in model design to meet new circumstances are an integral part of collaborative modelling. Section 5.1 details how SYMFOR makes use of formal Model Designs and Code Generator, Parameter Generator and Description Generator tools for model realisation. The sequence of activities for realising an individual model is however rather user-unfriendly. In the first stage users utilise a text-editor to create a new Model Design. They must be able to use the model specification language to do this. This requires that they understand the concepts used by the language *(i.e.* the language semantics) and that they use the correct syntax.

Diagram-based designs (*e.g.* those used by Stella, Powersim, ModelMaker, AME, described in Section 2.3.1) are superior in that they are easier to learn. Instead of having to learn vocabulary and syntax, users must simply familiarise themselves with a set of graphical symbols. Each of these symbols corresponds to a generic concept that can be used in an individual model (such as, for example, 'compartment'). They use a graphical user interface in which they can select, manipulate and customise these symbols to capture details of model design. The options available within the interface can be constrained to assist the user to build a well-formed model. For example, the interface may not allow users to connect an influence to a compartment.

One potential advantage of the text-based formal representation formalism used is that it is compatible with Prolog interpreters (as it conforms to Prolog syntax). This means that it should be relatively straightforward to exploit the special capabilities of Prolog to increase the functionality of SYMFOR. For example, it may be possible to construct tools to assist with model construction, to check the model for errors or to facilitate real-time interrogation of model structures (Muetzelfeldt *et al.*, 1989). In particular there may be scope for integrating SYMFOR and AME, as the latter system is also Prolog-based (R. Muetzelfeldt, Pers. Comm.). This means that in future it may be possible to combine or reproduce SYMFOR functionality (*e.g.* strong support for visualisation) and AME functionality (*e.g.* diagram-based model design) if both are developed in future.

The model realisation process is also sub-optimal because it is necessary to run two separate applications (the SYMFOR Model Compiler (SMM) and the Source-code compiler) to produce a runnable model on the basis of the design. Originally the intention was to invoke the source-code compiler automatically from the SMM. This is theoretically possible and, indeed, desirable. However, currently the source-code produced by the code-generator needs to be tweaked or debugged before it can be compiled. This is because of certain issues that the SMM cannot currently resolve. For example, with the C-compiler used, a program must be split up into separate sections contained in separate files (this is a legacy from the segmented memory architecture of 16-bit systems). For the SMM to be able to handle this it would need to be able to work out the total length of source code required and to be able to split up the source-code appropriately – a non-trivial task.

The requirement that SYMFOR should support reuse of model content (Requirement 11) is achieved by use of modularity. The approach taken is similar to that described in Muetzelfeldt (1995). In SYMFOR modules are stored and distributed in source-code form. One advantage of this is that any errors in their formulation can be more easily diagnosed. This is more difficult when modules are stored in binary form (*e.g.* as described in Smith, 1998). A disadvantage is that they must be compiled before they can be used. Also, they can only be altered by someone with a knowledge both of C programming and of the conventions for creating modules in SYMFOR. In some cases the creation of new modules can be trivial *e.g.* when one equation is substituted for another. In other cases it may be much more difficult to design a new module *e.g.* when creating a new module for reduced impact logging.

SYMFOR modules are algorithms are expressed in the C computer language. The use of C rather than a dedicated macro-language means that complex algorithms can be specified. These are required for some model features. For example, the algorithm required for specification of stand logging is necessarily complex. Any such algorithm must specify the selection of individual trees and their extraction from the stand. Extraction routes (skidtrails) are often designed to minimise total damage to the stand, and the module must contain sufficient functionality to be able to capture this. The strategy for modularity used in SYMFOR gives the expressiveness of C combined with the utility of using a higher-level specification language.

8.5 MODELLING LANGUAGE CONCEPTS

Requirement 12 specifies that logical grouping of model design elements should be supported. This achieved using the abstraction of a 'modelled entity' described in Section 4.3. A modelled entity groups data and algorithms used in the representation of a feature of the stand being modelled. For example, attribute values for stem diameter, height, diameter increment, crown-point are all logically associated with an individual tree.

The concept of a 'modelled entity type' satisfies Requirement 15. This is a set of modelled entities sharing the same attributes (but not necessarily the same attribute values). The concepts are very similar to those of class and instance in Object Oriented Modelling (as described by Rumbaugh *et al.*, 1991; Booch, 1994). However there are several differences. Modelled entity types cannot inherit slots from one another in the same way as classes can and also the concept of encapsulation is redundant (it is a programming issue).

Modelled entities can be created and destroyed, satisfying Requirement 13. This feature is useful when cohorts or individual trees represent trees in a stand (as usually a simulation run will involve creation and destruction of these kinds of representation units).

The framework draws on many concepts from System Dynamics (Requirement 16). Specifically the following concepts are used (see Section 4.3 for full details of use in SYMFOR and Bossel (1994) or Haefner (1996) for details of general usage):

- 'model state' - this is initialised at the start of individual simulation runs, and is modified in a series of iterations.
- 'state-variable' - this is a variable associated with a modelled entity that changes in a simulation run by increment or decrement.
- 'intermediate variable' - this is a variable that must be calculated every time-step (*i.e.* it is influenced by other model variables which themselves change through time). The previous value held by the variable is not taken into account when calculating a new value.
- 'parameter' - this refers to a value (or set of values) used in model calculations that is not associated with an individual modelled entity and that is not altered in the course of a simulation run (unless by a user).

Other concepts were introduced where the concepts described previously proved inadequate These are described in full in Section 4.3, but include:

- 'event' - this is a happening associated with individual modelled entities that has the potential to occur every iteration.

- 'classifier' - this is an item of data associated with an individual modelled entity that identifies which of a set of possible parameter values should be used in calculations pertaining to the modelled entity.

- 'modelled entity constant' - this is a value associated with an individual modelled entity which does not change for the lifetime of the modelled entity.

Subtle differences exist in the way that some of the terms are used in SYMFOR and the way that they are used in conventional system dynamics. First, state-variables and intermediate variables are always associated with modelled entities and cannot be free standing. Second, a distinction in SYMFOR is drawn between 'parameters' and 'modelled entity constants'. Both of these are equivalent to the conventional definition of parameters *i.e.* un-influenced influencers (*e.g.* Haefner, 1996). However, the two are used in quite a different way in that modelled entity constants change in number in the course of a simulation as modelled entities are created and destroyed while parameters are fixed.

One way in which the model design process could be improved is by the augmentation of modelling concepts to include geometrical concepts. At the moment in SYMFOR the shape of a modelled entity is captured using an informal convention whereby a set of ME constants correspond to vertices or points of the shape. This is time-consuming, involved and potentially unnecessary. For example, to define a shape with four vertices eight ME constants must be defined in the model design, one for each co-ordinate. If there was a construct then it might be possible to achieve the same result with a single predicate. This would also make it easier to develop special functionality. For example, if a standard representation of shapes is used then it becomes possible to write procedures for translation or transformation or determining if a point resides inside the shape.

8.6 USER INTERACTION

SYMFOR features a common interface for all models. This is Requirement 17 of those listed in Chapter 3. The interface is easy to use, (helping satisfy Requirement 18, that the framework should usable by people with a limited technical background). Users new to the software were able to load and run models very shortly after being introduced to the SYMFOR software. This was possible because of the extensive prototyping and user-trials of the software that were undertaken as part of the development process (Muetzelfeldt and Young, 1996, 1997). A lot of effort was made to elicit and conform with expectations and demands of framework users. For example, modellers expect to have to initialise state-variables and read in parameter values when using a model. Initially default files were automatically used each time a run was started. This meant that users did not see the files used unless they specifically sought to. This created confusion amongst users, so explicit selection of parameters and stand initialisation were made required activities at the start of every run. The handling of parameter files was also changed in response to demands from users. Parameter files were originally binary, and created by a special process. This generated complaints - users wanted to be able to enter parameter data and store alternative parameter sets using tools such as text editors. SYMFOR was therefore changed to handle text parameter files.

One problem with making the software easy to use is that advanced users may find some of the features cumbersome. For example, having to set the Stand Initialisation File and parameter set is time consuming at the start of simulation runs is cumbersome. The architecture of SYMFOR is such that these problems could be addressed by the creation of new shells to handle interaction with model executables. These new shells could act as replacements for the SYMFOR Model Manager (SMM), and offer features of use to more advanced users. This is possible because the shell and the model executable are separated.

The framework allows interactive simulation (Requirement 19). Combined with the visualisation described below, this is recognised as an important way in which the capabilities of simulation software can be extended (Bell and O'Keefe, 1987; Rooks, 1993). These capabilities allow models in the framework to be used in training activities - Awang (1994) discusses the need to make use of new technologies in forestry training. Use of a simulation model has several advantages over alternative field based experiments or analysis of existing data. By virtue of being interactive it is more stimulating. New simulation experiments can be created, and users can specify the collection of data relevant for testing a particular hypothesis. It is also useful for illustrating modelling principles. For example, simulation experiments proved highly effective in teaching of stochasticity. One advantage of having a system that supports multiple designs is that simple models (which run fast) can be used for teaching while more complicated models (which run more slowly) can be used for applications in which a higher quality of prediction is required.

Requirement 20 specifies that the framework should provide methods for meaningful display of results. This is achieved by the use of several displays (Section 6.2.1.4 – 6.2.1.8). These can be updated while models are running to create an animation-like effect. Each display can be customised by users to a high degree using a display options panel (Section 6.2.1.15 – 6.2.1.17). This means that they can easily be adapted to handle different model content.

Models in the system all have an associated Description file that contains a natural language description of the model content (Section 5.1.8). This description is accessible from the SMM (Section 6.2.1.3). This arrangement satisfies Requirement 21, that information that is comprehensible, unambiguous and complete should be available for all models in the system. Automatic description generation is superior to manual code generation because it is faster, and can deal with changes in model content that occur through time much more easily. Shortcomings in the manual method have been noted by Muetzelfeldt (1989) and Lorek *et al.* (1998).

8.7 CONCLUDING REMARKS

This thesis has described the development of a framework for modelling tropical forest dynamics. It has sought to demonstrate a number of things. First, I hope this thesis goes some way to making the case for a cross-disciplinary, scientific approach for addressing applied problems of management. Techniques and ideas from ecology, ecological modelling, forest management, computer science and artificial intelligence have been used in the development of SYMFOR. While this inevitably means that depth has had to be sacrificed in some areas, the approach means that valuable insights can be gained resulting in a superior design.

Second, I hope that a case has been made for adopting a more flexible approach to modelling using specialised tools such as modelling environments. Compared with the traditional approach of manually coding and re-coding models, modelling environments are fast, efficient and more powerful in that they are usable by people with a limited technical background.

Finally I hope I have proved that simulation modelling is relevant and useful for achieving the goal of sustainable forest management. Given the current threats to the forest, simulation modelling is too valuable a technique to be neglected lightly.

References

Abel, D.E. and Niven, B.S. (1990) Application of a formal specification language to animal ecology I: environment. Ecological Modelling, 50: 205-212.

Adlard, P.G., Spilsbury, M.J. and Whitmore, T.C. (1988) Current thinking on modelling Tropical Mosit Forest. Meeting of IUFRO. Sections s4.01.02 & 1.07. Growth and yield in mixed/moist forests. Forest Research Institute of Malaysia.

Ahmad, N. (1996) An assessment and enumerations of vesicular-arbuscular mycorrhiza propagules in some forest sites of Jengka. Journal of Tropical Forest Science, 9:137-146.

Alder, D. (1995) Growth Modelling for Mixed Tropical Forests. Tropical Forestry Papers No. 30. Oxford Forestry Institute, University of Oxford.

Alexander, I., Ahmad, N. and Lee, S.S. (1992) The role of mycorrhizas in the regeneration of some Malaysian forest trees. Philosophical Transactions of the Royal Society of London B, 335: 379-388.

Anon (1993) Tebang Pilih Tanam Indonesia (Indonesian selective cutting and planting) Keputusan Direktur Jenderal Pengusuhaan Hutan Nomor 151/KPTS/IV- BPHH/1993 Departmen Kehutanan, Jakarta.

Appanah, S. and Manaf, M.R.A. (1994) Fruiting and seedling survival of dipterocarps in a logged forest. Journal of Tropical Forest Science, 6: 215-222.

Appanah, S. and Rasol, A.M.M. (1995) Dipterocarp fruit dispersal and seedling distribution. Journal of Tropical Forest Science, 8: 258-263.

Argent, G., Campbell, E.J.F., Wilkie, P. and Saridan, A. (1993) Diversity in Experimental Plots, Wanariset Sangai, Central Kalimantan and work towards an Identification Manual for Economic Trees of Central Kalimantan, Unpublished manuscript.

Ashton, P.S. (1964) Ecological studies in the mixed Dipterocarp forests of Brunei state. Clarendon Press, Oxford.

Ashton, P.M.S. (1992) Some measurements of the microclimate within a Sri lankan tropical rainforest. Agricultural and Forest Meteorology, 59: 217-235.

Ashton, P.S. (1989) Dipterocarp reproductive biology. In: H. Leith and M.J.A. Werger, (Eds.).Tropical Rain Forest Ecosystems, pp. 219-240. Amsterdam: Elsevier.

Ashton, P.S. and Hall, P. (1992) Comparisons of structure among mixed dipterocarp forests of North Western Borneo. Journal of Ecology, 80: 459-481.

Assmann, E. (1970) The principles of forest yield study. Pergamon Press, Oxford.

Awang, K. (1994) Response to users' needs in forestry education with special reference to technical and socio-economic aspects of silviculture and management in the Asia-Pacific region. IN Forestry education: New trends and prospects. Food Agriculture Organisation of the United Nations, Rome, Italy.

Baharuddin, K., Mokhtaruddin, A. and Nik Muhamad, M. (1995) Surface runoff and soil loss from a skidtrail and a logging road in a tropical forest. Journal of Tropical Forest Science, 7: :558-569.

Baillie, I.C., Ashton, P.S., Court, M.N., Anderson, J.A.R., Fitzpatrick, E.A. and Tinsley, J. (1987) Site characteristics and the distribution of tree species in mixed dipterocarp forest on tertiary sediments in central sarawak, malaysia. Journal of Tropical Ecology, 3: 201-220.

Basnet, K. (1992) Effect of Topography on the Pattern of Trees in Tabonuco (Dacroydes excelsa) Dominated Rain Forest of Puerto Rico. Biotropica, 24: :21-42.

Baveco, J.M. and Smeulders, A.M.W. (1994) Objects for simulation: Smalltalk and Ecology. Simulation, 62 :42-57.

Bell, P.C. and O'Keefe, R.M. (1987) Visual interactive simulation history, recent developments, and major issues. Simulation, 49: 109-116.

Bertault, J.G. and Sist, P. (1995) The effects of logging in natural forests. Bois et Forêts des Tropiques.: 5-12.

Biging, G.S. and Dobbertin, M. (1992) A comparison of Distance Dependent Competition measures for Height and Basal Area Growth of Individual Conifer trees. Forest Science, 38: 695-720.

Biging, G.S. and Dobbertin, M. (1995) Evaluation of competition indices in individual tree growth models. Forest Science, 41: 360-377.

Booch, G. (1994) Object Oriented Analysis and Design with Applications. Second edition. The Benjamin/Cummings Publishing Company, California.

Bossel, H. (1994) Modeling and simulation. AK Peters, Wellesly, U.S.A.

Bossel, H.H. and Schaffer, H. (1989) Generic simulation model of foret growth, carbon and nitrogen dynamics, and application to tropical acacia and European spruce. Ecological Modelling, 48: 221-265.

Bossel, H. and Kreiger, H. (1991) Simulation model of natural tropical forest dynamics. Ecological Modelling, 9: 37-71.

Bossel, H. and Kreiger, H. (1994) Simulation of multi-species tropical forest dynamics using a vertically and horizontally structured model. Forest Ecology and Management, 69:123-144.

Botkin, D.B., Janak, J.F. and Wallis, J.R. (1972) Some ecological consequences of a computer model of forest growth. Journal of Ecology, 60: 849-872.

Brokaw, N.V.L. (1985) Treefalls, regrowth, and community structure in tropical forests. IN The Ecology of Natural Disturbance and Patch Dynamics. Eds Pickett, S.T.A. and White, P.S. pp 53-69. Academic Press, Orlando, Florida.

Brown, N. (1993) The implications of climate and gap microclimate for seedling growth conditions in a Bornean lowland rain forest. Journal of Tropical Ecology, 9: 153-168.

Brunjizeel, L.A. (1996) Predicting the hydrological impacts of land cover transformation in the humid tropics: the need for integrated research. In. J.H.C. Gash, (Ed.). 'Amazonian deforestation and climate.' Wiley, Chichester.

Budgen, D. (1994) Software Design. Addison-Wesley Publishing company, Wokingham, England.

Bugmann, H, Fischlin, A. and Kienast, F. (1996) Model convergence and state variable update in forest gap models. Ecological Modelling, 89: 197-208.

Cannon, C.H., Peart, D.R., Leighto, M. and Kartawinata, K. (1994) The structure of lowland rainforest after selective logging in West Kalimantan, Indonesia.

Carruthers, R.I., Larkin, T.S., Soper, R.S. (1988) Simulation of insect disease dynamics: an application of SERB to a rangeland ecosystem. Simulation, 52: 101-109.

Caswell, H. (1989) Matrix Population Models. Sinauer Associates, Inc. Sunderland Massachusetts.

Caswell, H., Koenig, H.E. and Resh, J.A. (1972) An introduction to systems science for ecologists. In: B.C. Patten, (Ed.). 'Systems Analysis and Simulation in Ecology. Vol II.' Academic Press, New York.

Chazdon, R.L. (1991) Sunflecks and their importance to forest understorey plants. Advances in Ecological Research, 18:1-56.

Chazdon, R.L., Pearcy, R.W., Lee, D.W. and Fetcher, N. (1996) Photosynthetic responses of tropical forest plants to contrasting light environments. In: S.S. Mulkey, R.L. Chazdon, and A.P. Smith (Eds.).Tropical forest plant ecophysiology, pp. 5-55. New York: Chapman and Hall.

Cherwell Scientific (1998) ModelMaker: One stop simulation and analysis. http:\\www.cherwell.com\modelmaker\

Clark, D.B. and Clark, D.A. (1996) Abudance, growth and mortality of very large trees in neotropical lowland rainforest. Forest Ecology and Management, 80: 235-244.

Claussen, J.W. (1995) Stem allometry in a North Queensland Tropical Rainforest. Biotropica, 27: 421-426.

Clearwater, M. (1996) Methods for the post-logging assessment of the ITFMP permanent sample plots. ITFMP Internal Report.

Dale, V.H., Doyle, T.W., Shugart, H.H. (1985) A comparison of tree growth models. Ecological Modelling, 29:145-169.

Dawkins, H.C. (1963) Crown diameters: their relation to bole diameter in tropical forest trees. Commonwealth Forestry Review, 42: 114: 318-333.

Davis L.S. and Johnson, K.N. (1987) Forest management. McGraw-Hill Inc. New York.

DeAngelis, D.L. and Gross, L.J. (1992) Individual-based models and approaches in ecology: populations communites and ecosystems. Chapman and Hall, New York.

De Fretes, Y. (1992) Community versus company-based rattan industry in Indonesia. In The Rainforest harvest: Sustainable Strategies for Saving the Tropical Forests? Friends of the Earth, London.

Deutschman, D.H., Levin, S.A., and Pacala, S.W. (1995) Seeing the forest for the trees: Community-wide predictions of a spatially explicit, individual-based mode of forest dynamics are insensitive to detail at the tree level. Bulletin of the Ecological Society of America 76(3 SUPPL):1995 320.

Ek, A.R., Monserud, R.A. (1974) FOREST: A computer model for simulating the growth and reproduction of mixed species forest stands. School of Natural Resources, University of Wisconsin, Research Report No. R2635 53pp.

Evans, J. (1992) Plantation forestry in the tropics. Oxford University Press.

Farquhar, A., Fikes, R., Pratt, W. and Rice, J. (1995) Collaborative ontology construction for information integration. Technical Report KSL-95-63, Stanford University, Knowledge Systems Laboratory.

Favrichon, V. (1996) Modelling the dynamics of a lowland Dipterocarp forest stand: Application of a density dependent matrix model. Anonymous CIRAD-Forets. :1-16.

Fox, J.E.D. (1972) The natural vegetation of Sabah and natural regeneration of the dipterocarp forests. PhD thesis, University of Wales.

Fox, J.E.D. (1973) Dipterocarp Seedling Behaviour in Sabah. Malaysian Forester, 36: 205-214.

Friend, A.D. and Schugart, H.H. (1993) A physiology-based gap model of forest dynamics. Ecology, 74: 792-797.

Garwood, N.C., Janos, D.P. and Brokaw, N. (1979) Earthquake caused landslides: a major disturbance to tropical forests. Science, 205:997-9.

Gennari, J.H., Oliver, D.E., Pratt, W. Rice, J. and Musen, M.A. (1995) A Web-Based Architecture for a Medical Vocabulary Server. Technical Report KSL-95-41, Stanford University, Knowledge Systems Laboratory.

Gitay, H. and Noble, I.R. (1997) What are functional types and how should we seek them? In 'Plant functional types: their relevance to ecosystem properties and global change' Cambridge University Press, Cambridge.

Goldammer, J.B. and Siebert, B. (1990) The impact of droughts and forest fires on tropical lowland rainforest of East Kalimantan. In J.G. Goldammer (Ed.), Fire in the Tropical Biota, pp. 11-31. Springer Verlag, Berlin.

Griersen Johns, A. (1997) Timber Production and Biodiversity Conservation in Tropical Rain forests. Cambridge University Press, Cambridge.

Gruber, T.R. (1993) A Translation Approach to Portable Ontology Specifications. Knowledge acquisition, 5: 199-220.

Gruber, T.R. and G. R. Olsen, G.R. (1994) An ontology for engineering mathematics. In J. Doyle, P. Torasso and E. Sandewall, Ed., Fourth International Conference on

Principles of Knowledge Representation and Reasoning, Gustav Stresemann Institut, Bonn, Germany, Morgan Kaufmann, 1994.

Haefner, J.W. (1996) Modeling biological systems: principles and applications. International Thomson Publishing, New York.

Hallé, F., Oldeman, R.A.A. and Tomlinson, P.B. Tropical trees and Forests: an architectural analysis. Springer-Verlag, Berlin.

Huston, M.A. (1994) Biological Diversity: The coexistence of species on changing landscapes. Cambridge University Press, Cambridge.

Huth, A., Ditzer, T. and Bossel, H. (1997) Rainforest Growth Model FORMIX3: A tool for forest management planning towards sustainability. Report to Deutsche Gesellschaft fur Technische Zusammenarbeit (GTZ).

Itoh, A., Yamakura, T., Ogino, K. and Lee, H.S. (1995) Survivorship and growth of seedlings of 4 dipterocarp species in a tropical rain-forest of Sarawak, East Malaysia. Ecological Research, 10:327-338.

Jackson, M.A. (1983) System Development. Prentice-Hall, New Jersey, USA.

Jarvis, P.G. (1993) Prospects for bottom up models. In: J.R. Elheringer and C.B. Field (Eds.). Scaling Physiological Processes: Leaf to Globe. Academic Press.

Johns, A.D. (1992) Species conservation in managed tropical forests. In: Whitmore, T.C. and Sayer, J. (Eds.). Tropical deforestation and Species Extinction. Chapman and Hall, London.

Johns, A.G. (1997) Timber production and biodiversity conservation in tropical rain forests. Cambridge University Press, Cambridge.

Jordan, C.F. (1989) Are process rates higher in tropical forest ecosystems? In: J. Proctor (Ed.). Mineral nutrients in tropical forest and savanna ecosystems. Blackwell Scientific, Cambridge.

Judson, O.P. (1994) The rise of the individual-based model in ecology. Trends in Ecology and Evolution, 9:9-14.

Kelley, A. and Pohl, I. (1990) A book on C: Programming in C. 2nd Edition. Benjamin/Cummings publishing company.

Kendall, K.E., Kendall, J.E., Systems Analysis and Design. Third Edition. Prentice Hall Inc, London.

Kennedy, D.N. (1991) The role of colonising species in the regeneration of dipterocarp rain forest. PhD Thesis, University of Aberdeen.

Kimmins, J.P. (1997) Forest ecology: a foundation for sustainable management. 2nd Edition. Prentice Hall, New Jerseys.

King, D.A. (1995) Allometry and life history of tropical trees. Journal of Tropical Ecology, 12: 25-44.

Köhler, P. and Huth, A. (1998) The effects of tree species grouping in tropical rainforest modelling: Simulations with the individual-based model FORMIND. Ecological Modelling,109: 301-321.

Korsgaard, S., (1989). The standtable projection simulation model. In: H.E. Burkhart, M. Rauscher and K. Johann (Eds.). Artificial intelligence and growth models for forest management decisions. Proc. IUFRO Meeting, Vienna, 18-22 Sept 1989. VPI&SU, Blacksburg VA, FWS-1-89

Knox,R.G., Kalb, V., Levine, E.R., Bindingnavle, U. (1998) A framework for Integrating Environmental Models to Simulated Forest Ecosystem Dynamics. http://sbg.ac.at/geo/idrisi/GIS_Environmental_Modelling/sf_papers/knox_robert/paper.html

Kurpick, P., Kurpick, U., Huth, A. (1997) The influence of Logging on a Malaysian Dipterocarp Rain Forest: a Study Using a Forest Gap Model. Journal of Theoretical Biology, 185:47-54.

Larkin, T.S., Carruthers, R.I., Soper, R.S. (1988) Simulation and object-oriented programming: the development of SERB. Simulation, 52: 93-100

Lawson, G.J., Crout, N.M.J., Levy, P.E., Mobbs, D.C., Wallace, J.S., Cannell, M.G.R. and Bradley, R.G. (1995) The tree crop interface: representation by coupling of forest and crop-process models. Agroforestry Systems, 30:199-221.

Lee, D.W., Bone, R.A., Tarsis, S.L. and Storch, D. (1990) Correlates of leaf optical properties in tropical forest sun and extreme shade plants. American Journal of Botany 77, 370-380.

Lee, D.W., Baskaran, K., Mansor, M., Mohamad, H. and Yap, S.K. (1996) Irradiance and spectral quality affect asian tropical rain-forest tree seedling development. Ecology 77, 568-580.

Levin, S.A., Cohen, D. and Hastings, A. (1984) Dispersal strategies in patchy environments. Theoretical Population Biology, 26: 165-191.

Lhotka, L. (1994) Implementations of individual-based models in aquatic ecology. Ecological Modelling, 74: 47-62.

Liew, T.C. and Wong, F.O. (1973) Density, recruitment, mortality and growth of Dipterocarp seedlings in virgin and logged-over forests in Sabah. Malaysian Forester, 36: 3-15.

Liu, J. and Ashton, P.S. (1997) Balancing biodiversity conservation and timber harvesting: An individual-based spatially explicit approach. Bulletin of the Ecological Society of America 77(3 SUPPL):1996268, 1997.

Lorek, H. and Sonnenschein, M. (1998) Object-oriented support for modelling and simulation of individual-oriented forest models. Ecological Modelling, 108:77-96.

Luan, J. (1994) Simulation of Forest Ecosystem Dynamics, with Respect to the Problem of Hierarchy. PhD Thesis, University of Edinburgh.

Lugo, A.E. and Scatena, F.N. (1996) Background and Catastrophic Tree Mortality in Tropical Moist, Wet and Rain forests. Biotropica, 28: 585-599.

Majid, N.M. and Jusoff, K. (1987) The impact of logging on soil temperature in a low elevation forest of Malaysia. Tropical Ecology, 28: 35-44.

Malmer, A. and Grip, H. (1990) Soil disturbance and loss of infiltrability caused by mechanical and manual extraction of tropical rainforest in Sabah, Malaysia.

Malpas, J. (1987) Prolog: A Relational Language and Its Applications. Prentice Hall International Ltd, London.

Maser, C. (1994) Sustainable Forestry: Philosophy, Science, and Economics. St. Lucie Press, Delray Beach, Florida.

Matthews, J.D. (1989) Silvicultural Systems. Clarendon Press, Oxford.

Maxwell, T. and Costanza, R. (1997) A language for modular spatio-temporal simulation. Ecological Modelling, 103: 105-113 1.

McCown, R.L., Hammer, G.L., Hargreaves, J.N.G., Holzworth, D.P., and Freebairn, D.M. (1994) APSIM: A novel software system for model development, model testing, and simulation in agricultural systems research. Agricultural Systems

Mendoza, G.G. and Setayarso, A., (1986) A transition matrix forest growth model for evaluating alternative harvesting schemes in Indonesia. Forest Ecology and Management, 15:219-228.

Mendoza, G.A. and Gumpal, E.C. (1987) Growth Projection of a Selectively Cut-Over Forest Based on Residual Inventory. Forest Ecology and Management, 20:253-263.

Minar, N., Burkhart, R., Langton, C. and Askenazi, M.(1996) The Swarm simulation system. A tool for building multi-agent simulations. http:\www.santafe

Mobbs, D.C. (1995) Coupling forest and crop models. In 'Annual report of Agroforestry modelling and research co-ordination'. ODA research programme R5652 and ITE Project T01065H1.

ModellData (1995) POWERSIM The Complete Software Tool for Dynamic Simulation. User Manual, ModellData AS. Helland, 5120 Manger, Norway.

Mooney, H.A. (1997) Ecosystem function of biodiversity: the basis of the viewpoint. In: T.M. Smith, H.H. Shugart and F.I. Woodward, (Eds.). 'Plant functional types: their relevance to ecosystem properties and global change' Cambridge University Press, Cambridge.

Muetzelfeldt, R., Robertson,D., Bundy,A. and Uschold, M. (1989) The use of Prolog for increasing the rigour and accessibility of ecological modelling. Ecological Modelling, 46:9-34.

Muetzelfeldt, R.I. (1995) A framework for a modular modelling approach in agroforestry. Agroforestry Systems, 30: 223-234.

Muetzelfeldt, R.I. and Sinclair, F.L. (1993) Ecological modelling of agroforestry systems. Agroforestry Abstracts, 6:207-247.

Muetzelfeldt, R.I. and Young, A.C., (1996) Report on SYMFOR presentations and workshops: 10 May - 3 June 1996. ITFMP Internal Report.

Muetzelfeldt, R.I. and Young, A.C. (1997) Report on workshop and handover of SYMFOR software (2-13 June 1997). ITFMP internal report.

Neches, R., Fikes, R., Timin, T., Gruber, T, Patil, Ramesh, Senator, T. and Swartout, W.R. (1991) Enabling technology for knowledge sharing. AI Magazine 12(3)

Nussbaum, R., Anderson, J., Spencer, T. (1995). Factors limiting the growth of indigenous tree seedlings planted on degraded forest soils in Sabah, Malaysia. Forest Ecology and Management, 74:149-159.

Oderwald, R.G. and Hans, R.P. (1993) Corroborating models with model properties. Forest Ecology and Management, 63:271-283.

O'Neill, R.V. and Rust, B.W. (1979) Aggregation error in ecological models. Ecological Modelling,7:91-105.

Osho, J.S.A. (1990) Matrix model for tree population simulation in a tropical rain forest of south-western Nigeria. Ecological Modelling, 59: 247-255.

Palmer, J. and Synnott. (1992) The management of natural forests. In: P. Sharmana (Ed.). Managing the World's forests - Looking For Balance Between Conservation and Development. Kendall Hunt Publishing Company, Iowa.

Petzold, C. (1992) Programming Windows 3.1. 3rd Edition. Microsoft Press, Washington.

Philip, M.S. (1994) Measuring trees and forests. 2nd edition. CAN International, Wallingford, Oxford.

Pickett, S.T.A. and White, P.S. (1985) The Ecology of Natural Disturbance and Patch Dynamics. Academic Press, Orlando, Florida.

Pinard, M.A. and Putz, F.E. (1996) Retaining forest biomass by reducing logging damage. Biotropica, 28: 278-295.

Pinard, M., Howlett, B., and Davidson, D. (1996) Site conditions limit pioneer tree recruitment after logging of dipterocarp forests in Sabah, Malaysia. Biotropica 28(1):2-12.

Poore, D. (1989) No timber without trees: sustainability in the tropical forest. Earthscan, London.

Primack,R.B., Lee, H.S., (1991). Population dynamics of pioneer (Macaranga) trees and understorey (Mallotus) trees (Euphorbiaceae) in primary and selectively logged Bornean rain forests. Journal of Tropical Ecology ,7:439-458.

Putz, F.E. (1984) The natural history of lianas on Barro Colorado Island. Ecology, 65: 1713-1724

Raharja, T. (1997) The role of modelling in producing forest management plans. MSc Dissertation, University of Edinburgh

Raich, J.W. and Christensen, N.L. (1989) Malaysian dipterocarp forest - tree seedling and sapling species composition and small-scale disturbance patterns. National Geographic Research 5:348-363.

Richards, P.W. (1952) The tropical rainforest: an ecological study. Cambridge University, Cambridge.

Riswan, S., Kenworthy, J.B., and Kartawinata, K. (1985) The estimation of temporal processes in tropical rain forest: a study of primary dipterocarp forest in Indonesia. Journal of Tropical Ecology, 1: 171-182.

Rooks, M. (1993) A user-centred paradigm for interactive simulation. Simulation, 60: 168-177.

Rothenberg, J. (1989) The nature of modelling. In: Artificial Intelligence, Simulation and Modelling. Widman, Koparo and Nielsen (Eds.). Publ. John Wiley and Sons.

Rumbaugh, J., Blaha, M., Permerlani, W., Eddy, F., Lorenson, W. (1991) Object Oriented Modeling and Design. Prentice-Hall Inc. New Jersey.

Running, S.W. and Coughlan, J.C. (1988) A general model of forest ecosystem processes for regional appliations. I. Hydrological balance, canopy gas exchange and primary production processes. Ecological Modelling,42:125-154.

Salwasser, H., MacCleery, D.W. and Snellgrove, A. (1993) An ecosystem perspective on sustainable forestry and directions for the U.S. National Forest System. In: G.H. Aplet, N. Johnson, J.T. Olson, and A. Sample (Eds.). 'Defining sustainable forestry' Island Press, Washington.

Scheffer, M., Baveco, J.M., DeAngelis, D.L., Rose, K.A., and Van Nes, E.H. (1995) Super-individuals a simple solution for modelling large populations on an individual basis. Ecological Modelling,80: 161-170.

Sessions, J. and Heinrich, R. (1993) Harvesting. In 'Tropical Forestry Handbook' Eds Pancel, L. Springer Verlag Berlin.

Shughart, H.H. (1984) A Theory of Forest Dynamics: The Ecological Implications of Forest Succession Models. Springer Verlag, New York.

Sianturi, P. (1996) Modelling competition between trees in tropical forests. PhD Thesis, University of Edinburgh.

Silva, J.N.M., de Carvalho, J.O.P., Lopes, J.C.A., Almeida, B.F., Costa, D.H.M., de Oliveira, L.C., Vanclay, J.K., and Skovsgaard, J.P. (1995) Growth and yield of a tropical rainforest in the Brazilian Amazon 13 years after logging. Forest Ecology and Management, 71: 267-274.

Smith, W.R. (1998) Model Reuse and Integration.
http://wally.usfs.auburn.edu/conference/IntroPaper.html

Sokal, R.R., Rohlf, F.J. (1995) Biometry: the principles and practice of statistics in biological research. Third edition. W.H. Freeman and Co., New York.

Stader, J. (1996) Results of the Enterprise Project. Report AIAI-TR-209, AIAI, University of Edinburgh.

Still, M.J. (1996) Rates of mortality and growth in three groups of dipterocarp seedlings in Sabah, Malaysia. In: M.D. Swaine (Ed.).The ecology of tropical forest tree seedlings, pp. 315-332. Paris/Carnforth: UNESCO/Parthenon.

Systems Thinking, Experimental Learning Laboratory (1994) STELLA II. Systems Thinking, Hanover. NH 05755.

Tamin, N (1992) The Utilization and Trade of Non-Timber Products in South-east Asia. In The Rainforest harvest: Sustainable Strategies for Saving the Tropical Forests? Friends of the Earth, London.

Turner, I.M. (1990) The seedling survivorship and growth of three Shorea spp. in a Malaysian tropical rain forest. Journal of Tropical Ecology 6, 469-478.

Urban, D.L., Bonan, G.B., Smith, T.M. and Shugart, H.H. (1991) Spatial applications of gap models. Forest Ecology and Management, 42: 95-110

Uschold, M., King, M., Moralee, S. and Zorgios, Y. (1998) The Enterprise Ontology. The Knowledge Engineering Review, Vol. 13, Special Issue on Putting Ontologies to Use (Eds. Mike Uschold and Austin Tate).

Uschold, M., Gruninger,M. (1996) Ontologies: Principles, Methods and Applications. Knowledge Engineering Review, 11(2)

Usher, M.B. (1966) A matrix approach to the management of renewable resources, with special reference to selection forests. Journal of Applied Ecology, 3:355-367

Van Der Meer, P.J., Bongers, F. (1996) Patterns of tree-fall and branch-fall in a tropical rain forest in French Guiana. Journal of Ecology, 84: 19-29.

Vanclay, J.K. (1994) Modelling Forest Growth and Yield: Applications to Mixed Tropical Forests. CAB International, Wallingford, Oxon.

Vanclay, J. K. (1995) Growth Models for Tropical Forests: A Synthesis of Models and Methods. Forest Science, 41: 7-42.

Wang, Y.P. and Jarvis, P.G. (1990) Description and validation of an array model - MAESTRO. Agriculture and .Forest .Meteorology, 51: 257-280

Weiner, J. W. and Thomas, S. C. Size variability and competition in plant monocultures. Oikos, 47:211-222.

Westoby, M. and Leishman, M. (1997) Categorizing plant species into functional types. In: T.M. Smith, H.H. Shugart and F.I. Woodward, (Eds.). 'Plant functional types: their relevance to ecosystem properties and global change' Cambridge University Press, Cambridge.

Whitmore, T. C. (1984) Tropical rain forests of the Far East. 2nd edition. Clarendon Press, Oxford.

Whitmore, T. C. (1989) An Introduction to Tropical Rainforests. Clarendon, Oxford

Whitmore, T.C. (1996) A review of some aspects of tropical rain forest seedling ecology with suggestions for further enquiry. In: M.D. Swaine (Ed.). The ecology of tropical forest tree seedlings, pp. 3-39. Paris/Carnforth: UNESCO/Parthenon.

Whitmore, T.C. and Brown, N.D. (1996) Dipterocarp seedling growth in rain-forest canopy gaps during 6 and a half years. Philosophical Transactions Of The Royal Society Of London Series B- Biological Sciences 351:1195-1203.

Wibowo, P.M. (1995) Modelling the growth and yield of lowland dipterocarp forest using SYMFOR 1.0. MSc Dissertation, University of Edinburgh.

Woods, P. (1989) Effects of Logging, Drought and Fire on Structure and Composition of Tropical Forests in Sabah, Malaysia. Biotropica, 21: 290-298.

Zipperlen, S.W. and Press, M.C. (1996) Photosynthesis in relation to growth and seedling ecology of two dipterocarp rain forest tree species. Journal of Ecology, 84:863-876.

The SYMFOR tropical forest modelling framework

A.C. YOUNG and R.I. MUETZELFELDT

Institute of Ecology and Resource Management, The University of Edinburgh, Mayfield Road, Edinburgh EH9 3JU
Allen.Young@ed.ac.uk

SUMMARY

Forecasting growth and yield of tropical forest under different management scenarios is an important part of any strategy for sustainable forest management. The SYMFOR modelling framework uses individual-based, tree position models to make predictions. The framework is both user-friendly and very flexible. It is possible to custom-build models for specified forest types and particular management options using the framework. A case study considering the effect of length of felling cycle is presented; this is intended to demonstrate the utility of the approach rather than to make precise estimates of growth and yield. The structure of the model used and the processes of calibrating and initialising it for use in the experiment are described. Some qualitative behaviour produced by the model is consistent with that observed in the field, but the figures for growth and yield are low compared with others in the literature. Suggestions on how future versions of the model could be improved and applied for sustainable forest management are included.

Keywords: felling cycle, growth and yield, individual-based model, tropical forest dynamics.

INTRODUCTION

Indonesia has an area of 1.03 M km^2 under indigenous forest (FAO 1997) and is now committed to the International Tropical Timber Organisation agreement of implementing sustainable management of all production forest by the year 2000. Large scale forest exploitation started comparatively recently in the 1970s. For this reason there is a shortage of data (and hence uncertainty) concerning the long term sustainability of current management practices. Growth and yield modelling is an important tool to address this problem.

Tropical forests managed under polycyclic selective logging systems present particular difficulties for growth and yield modelling. Tropical forests have complex structures and show more small-scale spatial variation than plantations. These factors mean that many techniques suitable for the modelling of monospecific plantations are not well suited to modelling mixed tropical forests (Vanclay 1994).

Many different approaches have been applied to modelling the growth and yield of tropical forests. These include transition matrix modelling (Mendoza and Setayarso 1986), cohort modelling (Korsgaard 1984) and more complicated spatially-disaggregated size-class modelling (Bossel and Kreiger 1991). The utility of these approaches can be assessed using the criteria of precision, accuracy and generality (Vanclay 1995).

Tree position models (as defined by Alder 1995) are a class of *individual based model* (IBMs) that require co-ordinate data for each tree above a certain size threshold in a stand. Ek and Monserud (1974) provide an example of a tree position model that has been developed for use in temperate forests. The requirement for spatial information means that the models need more data than other types of model but IBMs can potentially give greater generality while being at least equal to the others in terms of precision and accuracy. The reason that this is so is that factors which affect the dynamics of a tropical forest often operate over small spatial scales and so are *local* to particular areas. For example, both tree growth and stand disturbance following treefalls are determined by local (rather than stand-level) factors. Tree growth is dependent upon the magnitude of competition for light and other resources with other trees in the immediate vicinity. The disturbance created by a treefall depends upon the characteristics of the falling tree and those of trees close to it.

If co-ordinate information is available then knowledge of the mechanisms of forest dynamics can be used to predict what will happen when the system is manipulated. For example, data on the stand disturbance created by single or a small number of treefalls can be used in a spatial approach to estimate the stand disturbance created by felling many trees at once (provided that it is possible to make assumptions concerning direction of treefall). Under heavy logging the areas affected by different treefalls are more likely to overlap. This means that stand-level disturbance cannot be derived by simply multiplying individual tree disturbance by the number of treefalls. For this reason a non-spatial stand-level model must be recalibrated for different logging intensities. Judson (1994) provides a more theoretical discussion on how IBMs may give greater generality.

Gap models are another class of IBM that have been used extensively, primarily in investigations of succession in forest ecosystems (Shugart 1984). In these models detailed

information on the location of individual trees is not used so that they are less able to capture the localised nature of individual interaction.

There is now a growing recognition that in some cases a single silvicultural system is likely to be unsustainable if applied rigidly over a large area or for a long period of time. This is because of spatial variability in forest, changes in environmental factors (*e.g.* nitrogen inputs) and also because of changes in the products demanded from forests by society over time (Maser 1994). For these reasons a system capable of evaluating *alternative* silvicultural systems is needed. The advantage of using a class of models with high generality, such as tree position models, is that they require little recalibration to investigate alternative systems.

The SYMFOR modelling system has been developed to provide estimates of growth and yield of a diverse range of forest types under alternative management regimes. SYMFOR is a modelling environment specialising in individual-based, tree-position models. The purpose of this paper is to describe briefly the environment and to illustrate how it can be applied to a particular problem (estimating growth and yield of forest with different cutting cycle lengths). No strong claims concerning accuracy of the results are made at this stage, though it is hoped that the potential of the approach will become apparent.

DESCRIPTION OF THE MODELLING FRAMEWORK

The current version of the SYMFOR software will run on any computer which can support Microsoft Windows™ 3.1 or MS Windows™ 95. The software has two components:

- the Model Designer, used to create new models;
- the Model Manager, used to run the models created with the Model Designer and to implement simulation experiments.

The process of designing a model in SYMFOR consists of defining a model *specification*, which completely describes the structure of a model in a formal and concise way. The specification holds details of the entities represented in the model and their attributes. For example, the specification might state that individual trees greater than 10 cm in stem diameter are represented in a model, and that each of these has attributes of stem diameter, total height, crown-point (defined as the height at which tree foliage first occurs) and the species group that it belongs to. Muetzelfeldt *et al.* (1989) discuss the theory and use of model specifications.

An important role of the specification is to identify the particular mathematical relationships (called modules) that are used in the model. Most types of entity have module *slots* (*i.e.* attributes for which a mathematical function is required). For example, trees may have a slot used to calculate tree height. Three alternative equations for calculating height may be represented in the modules height1, height2 and height3 which respectively contain, say, linear, quadratic and hyperbolic functions for calculating height on the basis of tree diameter. These modules are stored in the module library, and the person designing a particular model chooses one of theses modules when specifying the model.

One of the most important advantages of using model specifications is that it simplifies the process of designing and building models, while the main advantage of modularity is that it avoids unnecessary duplication of effort. Once a module has been built and put into the module library it is available for use in every SYMFOR model, and need not ever be recreated. There is no limit to the number of modules that can be stored in the library.

Once the model specification has been completed, the Model Designer is used to generate C source code which is then compiled and linked to produce a Windows program that can be loaded by the Model Manager. The Model Manager can then be used perform simulation runs of the model. Each simulation run commences with the user selecting a Stand Initialisation File (SIF). This describes the state of the stand at the start of a simulation in terms of the position, size and species of all trees. The file is generated from permanent sample plot data and can represent plots of different dimensions and areas (though usually one hectare plots are used).

The Model Manager has a number of displays that present information on the stand in a meaningful and accessible way. Each of these can be configured to suit the preferences of the user. The Model Manager also has a Parameter Editor that allows users to alter the value of the parameters used by the model in its calculations. Parameters are used as coefficients in model equations and also to specify management operations.

Simulation experiments can be conducted by repeatedly running the model with the Model Manager. Replicate runs of each combination of plot and treatment must be performed because of the stochastic nature of many of the SYMFOR models. The amount of replication used for a particular model will depend upon the variability obtained, and this depends upon the model specification.

CASE STUDY

Aim and rationale

The aim of the case study was to determine the effect of different cutting cycle lengths on productivity of stands of lowland dipterocarp forest. The length of cutting cycle is one of the most important factors under the control of the forest manager working with selective cutting systems. The Indonesian selective cutting and replanting system (TPTI) (Anon. 1993) has the length of the cutting cycle set at 35 years, normally with a diameter limit of 50 cm DBH for selected commercial species. In this system it is assumed that the trees forming the next crop remain after the harvest, allowing a relatively short cutting cycle. In the absence of long-term empirical data there is a need to employ models which capture ecological processes to evaluate the sustainability of the current system.

Procedure

Two data sets collected by the Forest Research Institute of Samarinda were analysed to determine parameter values. These were from plots at Wanariset Sangai Research forest, Central Kalimantan (described below) and those established in Berau region of East Kalimantan. The data from the Berau project were used to calculate growth parameters as this dataset had post-logging data of 17 years so that a meaningful analysis of data was more likely. Some coefficients were obtained by applying statistical curve-fitting techniques to the data. Some coefficients were obtained directly from the literature whilst others were given arbitrary values (for example the values that specify the point on the perimeter to which felled trees are skidded). Tables 1, 2 and 3 describe the parameters used in the model.

Data from three 1 hectare plots (100×100m) of primary forest at the Wanariset Sangai research forest Central Kalimantan (Indonesian Borneo, 1°29'S, 112°31'E) were obtained and used to initialise the model. This part of Borneo receives annual rainfall of over 3000 mm and has an average daily temperature of 25°C. Plots 1 and 7 are in valleys and have a stream running through them. Plot 5 is on a steep slope of 20-40° (Proctor 1994) and is noticeably more heavily stocked with intermediate-sized trees than the others.

SYMFOR release 2.21 was used in the case study. The software was mounted on an IBM compatible computer with a 100MHz Pentium processor and 16MB of RAM running Microsoft Windows™ 95.

Ten replicate runs were performed for each combination of plot and treatment. The length of each simulation was 160 years giving 4, 3 and 2 cutting cycles for three cycle lengths (35, 50 and 70 years). The statistic of Mean Annual Harvest (MAH) was calculated for each plot - treatment combination using the formula given in Equation 1.

$$MAH = \sum_{i=2}^{i=n} V \; / \; n.c \qquad \text{Equation 1}$$

where MAH is the Mean Annual Harvest in $m^3 ha^{-1} yr^{-1}$, i is the harvest number, n is the number of harvests in 160 years, V is the timber volume extracted in harvest i in $m^3 ha^{-1}$ and c is the cycle length in years. This is a similar statistic to Mean Annual Increment except that it uses harvested volume (rather than standing volume) and the first harvest is excluded from this analysis, as it is not representative of the management of logged-over forest. The treatments were compared using analysis of variance of the Mean Annual Harvest using a complete randomised block design (Sokal and Rolf 1992).

Model design

The model used in the case study had a time-step of one year. Parameters in the model were either not disaggregated or disaggregated by species-group and/or size-class. Six species-groups (Table 4) and four size-classes of tree (10 - 30, 30 - 50, 50 - 70 and > 70 cm DBH) were used.

Individual trees of stem diameter at breast height (DBH) greater than 10 cm were represented. Tree position, stem diameter and species-group of individuals were represented and variables of tree height, crown-point, crown radius, basal area, volume, shading-index and stem diameter increment were calculated annually.

The relationship between tree height and diameter was described by a non-rectangular hyperbola. Crown-point was represented as a linear function of height, while crown radius was modelled as a linear function of stem diameter. Basal area was calculated by assuming that the stem was circular in cross-section. Merchantable stem volume was calculated by multiplying the volume of a cylinder of diameter equal to the stem diameter and height equal to the crown-point minus the height of cutting by a form-factor. Individual values of basal area and merchantable volume for individual trees were summed to give estimates for the stand.

Diameter increment was modelled as a function of a shading index calculated for each tree. Hegyi's shading index (Dale, Doyle and Shugart 1985) was used to describe the amount of competition for light and other resources with other trees. It was calculated as the sum of the distance-weighted size ratio for every neighbouring tree which is equal to relative size of the neighbour (defined in terms of stem diameter) multiplied by the inverse of the distance between the neighbour and the tree. The relationship used to calculate diameter increment is given in Equation 2.

$$\Delta D_{stem} = k + 10^{-gmI} \qquad \text{Equation 2}$$

where ΔD_{stem} is the stem diameter increment of the tree in cm yr^{-1}, k is a coefficient that controls the asymptotic value of ΔD_{stem}, g is a coefficient that determines the sensitivity of ΔD_{stem} to increasing values of I, and I is the value of Hegyi's index.

The model used the concept of a *recruitment grid* to capture recruitment of individual trees to the individual-based submodel. The grid consists of a set of gridsquares, each 10 x 10 m, covering the entire modelled area. Leaf area index (LAI) defined as the total leaf area per unit ground area was calculated annually for each gridsquare. To do this the foliage area of each tree within the gridsquare was calculated as a function of basal area using a non-rectangular hyperbola. A weighted average of gridsquare foliage and the foliage of the eight surrounding gridsquares was then determined, and this was divided by area of the nine gridsquares to give gridsquare LAI. The model used a wrap-around technique to minimise edge effects when determining the gridsquares used in the weighted average. Each edge was mapped onto the edge that was parallel and opposite to it.

Each gridsquare had one associated *seedling-sapling cohort* (hereafter abbreviated to *cohort*) for each species-group in the model. Each of these represented a set of seedlings or saplings that had become established at the same point in time within the gridsquare. Each cohort had a state variable, *stage*, that described the developmental stage of the cohort in years and a variable *stage increment* to describe the annual change in the value of *stage*. Stage increment only occurred if gridsquare LAI was within a certain range of values. New individuals of 10cm stem diameter were created in the gridsquare when the *stage* of the cohort reached a critical level.

TABLE 1 *Parameters used in the model which have only one value. WS stands for Wanariset Sangai*

Parameter group	Parameter name	Source	Units	Value
Tree allometry	k - the ratio of crown radius to stem radius	Analysis of WS data	m cm^{-1}	25
	f - the proportion of a cylinder (with diameter equal to stem diameter and height equal to the tree crown-point minus the height of cutting) that constitutes the merchantable volume of the tree.	Analysis of WS data	-	0.45
	u - a coefficient giving the initial slope of the curve describing the relationship between tree foliage area and stem basal area.	Estimate	m^2 m^{-2}	800
	v - the maximum value of foliage area that a tree can have	Estimate	m^2	400
Tree growth	r_{nbr} - the range within which trees are considered to be competing with one another	Sensitivity analysis	m	15
Disturbance	N - the number of points within a gridsquare that are sampled to determine gridsquare disturbance	Sensitivity analysis	-	4
	s_0 - the relative size (based on stem diameter) of a tree in a potential damage zone below which no damage (*i.e.* mortality) occurs	Estimate	cm cm^{-1}	1.1
	s_1 - the relative size (based on stem diameter) of a tree in a potential damage zone above which damage (*i.e.* mortality) is certain	Estimate	cm cm^{-1}	3
Harvesting	x_{access} - the x-coordinate of the point on the perimeter where trees are dragged out of the stand	Arbitrary value	m	0
	y_{access} - the y-coordinate of the point on the perimeter where trees are dragged out of the stand	Arbitrary value	m	50
	t_{first} - the number of years after the start before the first logging takes place	Arbitrary value	years	5
	t_{cycle} - the number of years between subsequent loggings	Treatment	years	35, 50 & 70

TABLE 2 *Parameters that are disaggregated by species. These parameters have one value for every species group found in the model. Species groups are defined in Table 4*

Parameter group	Parameter	Source	Units	Value for each species-group					
				1	2	3	4	5	6
Tree allometry	a - the ratio of crown-point to total height	Analysis of data from Wanariset Sangai	m m^{-1}	0.55	0.55	0.55	0.55	0.55	0.55
	o - the maximum height that a tree can attain	Literature	m	60	70	70	85	70	50
	e - the initial slope of the curve relating tree height to diameter	Estimate	m cm^{-1}	200	200	200	200	200	400
Cohort	lai_0 - the lai below which there is no growth of the cohort	Estimate	m^2 m^{-2}	2.5	2.5	2.5	2.0	2.5	0.0
	lai_1 - the lai above which there is no growth of the cohort	Estimate	m^2 m^{-2}	8.0	6.0	6.0	6.0	6.0	2.0
	$T_{maturity}$ - the number of years a cohort must grow for before trees reach 10 cm stem diameter	Estimate	years	21	16	16	16	14	9
	E - the number of trees that a cohort will produce on attaining maturity	Estimate	-	0.4	0.4	0.4	0.4	0.4	0.6
Harvesting	d_{crit} - the critical stem diameter for logging	Literature	cm	70	70	70	70	70	NA

TABLE 3 *Parameters that are disaggregated by species and by size class. These parameters have one value for every combination of species group and size class. Species groups are defined in Table 4*

Parameter group	Parameter	Source	Size class (cm)	Species group 1	2	3	4	5	6
Tree growth	g - coefficient used in Equation 2	Analysis of Berau data	10-30	-0.35	-0.15	-0.15	-0.16	-0.20	-0.40
			30-50	-0.35	-0.15	-0.25	-0.16	-0.20	-0.40
			50-70	-0.40	-0.15	-0.25	-0.16	-0.20	-0.40
			>70	-0.40	-0.15	-0.25	-0.16	-0.20	-0.40
	k - coefficient used in Equation 2	Analysis of Berau data	10-30	0.75	0.80	0.80	1.00	0.50	1.40
			30-50	0.85	0.60	0.95	0.85	0.30	0.10
			50-70	0.85	0.50	0.70	0.70	0.30	0.10
			>70	0.75	0.50	0.00	0.00	0.20	0.00
Natural disturbance rate	P_{int} - the probability that an individual tree will fall over and initiate a disturbance event in a year	Estimate	10-30	0.13	0.13	0.13	0.13	0.13	0.04
			30-50	0.07	0.07	0.07	0.07	0.07	0.04
			50-70	0.07	0.07	0.07	0.07	0.07	0.04
			>70	0.07	0.07	0.07	0.07	0.07	0.04

TABLE 4 *Species groups used in model simulations. Six different species groups were used in the model. Note that some congeneric species occur in different groups - in these cases the timber group is given in brackets to indicate the set of species.* Shorea *and* Hopea *both contribute species to more than one group*

Abbreviation	Charateristics	Genera	Rationale
1. Heavy Dipterocarps	Slow growing: growth unresponsive to increased light: medium sized trees: long-lived:	*Shorea* (Balau) *Hopea* (Giam) *Vatica* (Resak) *Cotylelobium* (Resak)	Growth characteristics different from other dipterocarps: Grouping is the same as the commercial grouping of 'Heavy Hardwoods' recognised by foresters
2. Keruing	Medium growth rates:	*Dipterocarpus* (Keruing)	Growth characteristics intermediate for dipterocarps: Regeneration characteristics different than from other Medium dipterocarps: Grouping recognised by foresters:
3. Medium Dipterocarps		*Hopea* (Merawan) *Dryobalanops* (Kapur)	Growth characteristics intermediate for dipterocarps:
4. Light Dipterocarps	Large trees: fast growing: long-lived:	*Shorea* (Meranti) *Parashorea* (white Seraya) *Anisoptera* (Mersawa)	
5. Non Dipterocarps	Various	Non-dipterocarps that are not pioneers	Many species in this group have not been well enough studied to allow meaningful distinctions to be drawn:
6. Pioneers	Aggressive growth response to increased light: Relatively *short-lived*: Short in stature:	*Macaranga*: *Mallotus*: *Anthocephalus*:	Non-dipterocarps with obvious pioneer tendencies:

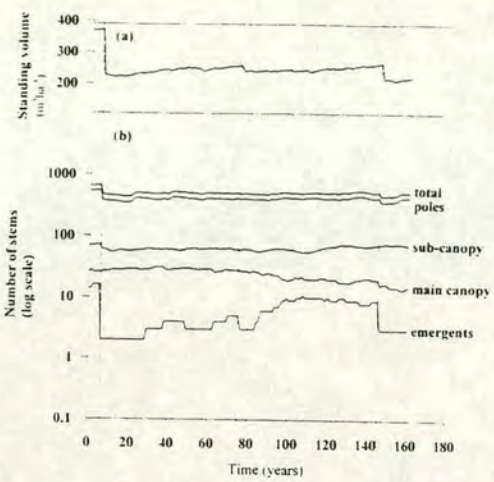


FIGURE 1 *Results from a single run performed as part of the investigation. The graphs show different stand characteristics through time. The dashed lines indicate years in which logging occurs. The run was initialised with data from Plot 5 of Wanariset Sangai Research forest. The stem diameter classes are: emergents (>70 cm); main-canopy (50-70 cm); sub-canopy (30-50 cm); poles (10-30 cm)*

Each model simulation included one or more harvests in which the felling and extraction of selected trees were simulated. All trees of commercial species groups above a critical stem diameter were selected. A skidtrail was created for every felled tree. These joined the base of the tree to a point on the perimeter of the stand in a straight line. The point was assumed to be the same for all extracted trees.

Stand disturbance events were initiated by treefalls (either natural or harvesting) and by skidding of felled trees from the stand. Natural disturbance was modelled by assuming that each tree had a certain probability of initiating an event in a year. The probabilities were the same for all individuals of the same species-group.

Disturbance events affected nearby individual trees greater than 10 cm stem diameter and nearby cohorts in the stand. Each object initiating the disturbance was projected onto the recruitment grid to define the *potential damage zone* (pdz) of the object. Individual trees inside the pdz each had a probability of mortality in the year in which the event occurred. In a skidtrail pdz this probability was 1 for all trees below a critical size. In a treefall pdz this probability was a function of the relative size (the diameter of the pdz tree divided by diameter of the initiator).

Destruction of cohorts occurred when the disturbance in the gridsquare with which the cohort was associated was greater than a critical value. To determine gridsquare disturbance a series of randomly located points was generated inside each square and the number of these points inside one or more pdzs was determined. This was divided by the total number of points in the gridsquare to give gridsquare disturbance. The stand disturbance index was defined as the sum of the disturbance index for all gridsquares.

Results

The graphs contained in Figure 1 show how selected stand characteristics change during a run in which the model was initialised with Plot 5 and a felling cycle set for 70 years was simulated. Harvests occurred in years 5, 75 and 145. Figure 1 (a) shows the standing volume of the stand through time. There is a reduction from 370 m^3 to 220 m^3 at the first harvest and most of this was accounted for in timber of the big trees that were removed. Subsequent harvests yielded much less timber, of the order of 10m^3 yr^{-1}. The first harvest is much higher than that typically removed in Indonesia under TPTI (Bertault and Sist 1995). This discrepancy may arise for reasons discussed in the next section. (The discrepancy is actually useful in that one role of any modelling is to identify areas where future research and development may prove beneficial).

The number of standing trees in different size classes through time is shown in Figure 1 (b). Apart from the harvested trees, harvesting has greatest effect on the poles in the stand with for example about 140 poles (21 % of the total number of trees) being lost in the first harvest. Sub-canopy and main canopy trees show much less variation with time. Most of the emergents are removed in the first harvest, but numbers recover by around year 120 of the simulation. The number of emergents decreases significantly again after the third harvest in year 145 of the simulation.

Table 5 shows the MAH rates obtained for plots with different felling cycles. Analyses of variance undertaken on MAH showed Plot factor to be highly significant ($p < 0.0001$) while the length of felling cycle was not significant ($p = 0.1743$). Plot 5 had the highest MAH for all three treatments. As this plot was the one best stocked with intermediate-sized trees, the model suggests that initial stocking is more important than treatment in determining yield.

TABLE 5 *Mean Annual Harvest rates stimulated for different plot felling cycle combinations. Each cell contains the mean and the standard error (n = 10)*

m^3 ha

	35 year felling cycle	50 year felling cycle	70 year felling cycle
Plot 1	0.31 ± 0.018	0.29 ± 0.024	0.31 ± 0.027
Plot 5	0.54 ± 0.049	0.52 ± 0.031	0.45 ± 0.028
Plot 7	0.38 ± 0.025	0.31 ± 0.024	0.34 ± 0.023

Possible improvements to model design

The model design was constrained by several factors, mainly those of data availability and mathematical intractability (in the case of topography/skidtrail design). The most important improvements to the design are likely to be:

- Representation of plot topography and inclusion of realistic skidtrail design. Plot topography may influence the growth and mortality rates of trees, the selection of trees for extraction and planning of skid trails in harvesting operations (Pinard and Putz 1996) and rates of soil erosion following canopy opening (Bahruddin *et al.* 1995).
- Inclusion of nutrient dynamics. Currently nutrient dynamics are not modelled, mainly because there is a lack of data that would be required to calibrate suitable relationships. However real forests may undergo progressive impoverishment in nutrients as a result of silvicultural operations. This may lower the growth rates of trees and hence the productivity of forest stands.
- Inclusion of more detailed representation of tree morphology. At the moment a single state-variable (DBH) is used to capture morphological development, so that trees of the same DBH and species group always show the same form. However trees growing in different environments often show different developmental patterns (*e.g.* King 1995).

DISCUSSION

Use of environments for forest modelling

SYMFOR is a modelling environment specialising in individual-based models, rather than a single individual based model. This means that it is flexible enough to cope with modelling different forest stands and different management operations. For example, in some stands the local height of the water table within the stand may influence the growth rates of trees, while other stands may drain freely. If a single model were to be used then it would have to include water dynamics to be able to cope with the former case. This would lower its suitability for use in the second case: irrelevant data would have to be collected, the model would be more complex than need be (and so more difficult to understand) and the model would also run more slowly. Use of modelling environments in which models can readily be adapted and new models can be created is therefore an important part of any large-scale modelling activity.

Stochasticity in individual-based models

Figure 2 shows the high amount of variability among replicate runs caused by stochasticity in processes such as disturbance in the model. Other models, such as size-class models, may produce behaviour that varies less between replicate runs. However they may achieve this by suppressing behaviour associated with the largest trees in the stand. This is because aggregation of individuals will usually produce bias towards median individuals and away from more extreme individuals. There are likely to be few large trees in a stand so that often the size-class used to contain them in a size-class model will be wide. Moreover the behaviour of the largest trees is disproportionately important – they contain a relatively large proportion of standing volume of the stand (which will be immediately lost when they fall) and they also are more likely to dislodge other trees when they fall (Clark and Clark 1996). The result of aggregation may be that size-class models systematically underestimate factors such as stand disturbance and the variability that would occur in real stand.

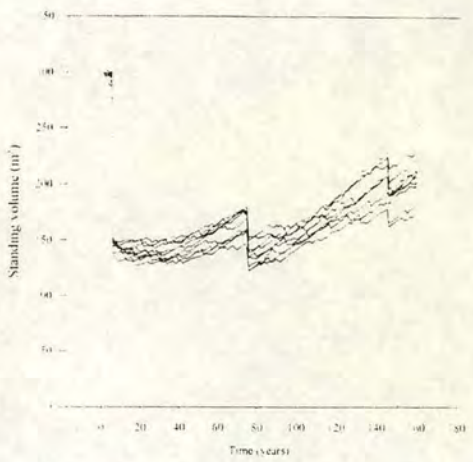


FIGURE 2 *Variability between replicate runs. The graph shows how stand volume develops through time for different runs which are initialised with the same stand and use the same set of parameter values. The variability is due to the presence of stochastic elements in the model.*

Field-based empirical approaches also suffer from high variability caused by irregularly timed and rare events such as the fall of large trees. They compare unfavourably with modelling approaches in that replication to assess sensitivity of results to these events involves establishing new plots. In both modelling and purely empirical approaches variability can be reduced by increasing plot size to greater than 1 ha, *e.g.* to 4 ha. In the case of an IBM approach this has also the advantage of lessening edge effect. The alternative approach to reduction of edge effect, that of using wrap-around techniques, has the difficulty that it will tend to bias results by exaggerating the influence of edge phenomena.

Implications for data collection

The main differences between the IBM approach described and its alternatives are a commitment to model individuals and to capture local interactions within the stand. Both have consequences for data collection.

Model initialisation requires information on individual trees of the stand, specifically their diameter, species-group and position within the stand. Given that the sampling unit of most mensurations is the individual tree, most of this information is available from previously established plots. The exception to this is tree position, which is infrequently

collected. However, techniques already exist for modelling the spatial distribution of individuals at a single point in time (*e.g.* Ripley 1987) and it is possible that some of these could be adapted for use in the initialisation of IBMs. Also it is possible to collect position data retrospectively, so that it may be possible to obtain reasonable data sets from recently established plots in which mortality has not been excessive.

Calibration can be done with sample rather than census data. Standard techniques can be employed for calibration of allometric and growth relationships for individual trees. The purpose of the latter calibration is to remove as much bias as possible rather than to enable precise predictions of individual tree growth (which are very difficult to achieve).

In the model all mortality occurs in relation to disturbance events. This leads to two main differences with respect to data collection. First, the roles of different trees that fall over as part of the same event must be distinguished, that is the trees responsible for initiating the event must be distinguished from those that are killed as a result of the action of the initiators. Second, all fallen trees must be mapped. Potential damage zones must be delimited for each initiator and an attempt made to relate the characteristics of these zones to the characteristics of the initiator.

Some of the calibration requirements are for time series data which can only come from Permanent Sample Plots. Natural disturbance rates and growth rates of individual trees are most reliably determined using a set of measurements repeated over time from the same plots. If the natural disturbance that occurred in the previous year can be reliably distinguished from older natural disturbance events then it may even be possible to relax the requirement for time series data in this case. Nearly all other calibration and all initialisation requirements can be met with data from Temporary Sample Plots.

ACKNOWLEDGEMENTS

This work was undertaken as part of the Indonesia-UK Tropical Forest Management Programme which was funded by the Department for International Development of the United Kingdom and the Ministry of Forestry of the Government of Indonesia. The authors wish to acknowledge the contribution of Paul van Gardingen who read the manuscript and made helpful suggestions. the Forest Research Institute of Samarinda, Indonesia who collected and provided data sets for analysis and staff of the EU funded Berau Forest Management Project who also provided assistance.

REFERENCES

ALDER, D. 1995 Growth modelling for mixed tropical forests. *Tropical Forestry Papers* No. 30. Oxford Forestry Institute, University of Oxford.

ANON. 1993 Tebang Pilih Tanam Indonesia (Indonesian selective cutting and planting). Keputusan Direktur Jenderal Pengusuhaan Hutan Nomor 151/KPTS/IV - BPHH/1993. Departemen Kehutanan, Jakarta.

BAHARUDDUN, K., MOKHTARUDDIN, A. and NIK MUHAMAD, M. 1995 Surface runoff and soil loss from a skidtrail and a logging road in a tropical forest. *Journal of Tropical Forest Science* **7**(4):558-569.

BERTAULT, J.G. and SIST, P. 1995 The effects of logging in natural forests. *Bois et Forêts des Tropiques*: 5-12.

BOSSEL, H. and KRIEGER, H. 1991 Simulation model of natural tropical forest dynamics. *Ecological Modelling* **9**: 37-71.

CLARK, D.B. and CLARK, D.A. 1996 Abundance, growth and mortality of very large trees in neotropical lowland rainforest. *For. Ecol. and Manage.* **80** (3): 235-244.

DALE,V.H., DOYLE,T.W. and SHUGART, H.H. 1985 A comparison of tree growth models. *Ecological Modelling* **29**: 145-169.

EK, A.R. and MONSERUD, R.A. 1974 FOREST: a computer model for simulating the growth and reproduction of mixed species forest stands. Research Report R 2635, School of Natural Resources, College of Agricultural and Life Sciences, University of Wisconsin - Madison.

FAO. 1997 State of the world's forests. FAO, Rome.

JUDSON, O.P. 1994 The rise of the individual-based model in ecology. *Trends in Ecology and Evolution* **9**: 9-14.

KING, D.A. 1995 Allometry and life history of tropical trees. *Journal of Tropical Ecology* **12**:25-44.

KORSGAARD, S. 1989 The standtable projection simulation model. *In:* BURKHART, H.E., RAUSCHER, M. and JOHANN, K. (eds.) *Artificial intelligence and growth models for forest management decisions*. Proc. IUFRO Meeting, Vienna, 18-22 Sept 1989. Publ. FWS-1-89, VPI&SU, Blacksburg VA, USA.

MASER, C. 1994 *Sustainable forestry: philosophy, science, and economics*. St. Lucie Press, Delray Beach, Florida.

MENDOZA, G.G. and SETAYARSO, A. 1986 A transition matrix forest growth model for evaluating alternative harvesting schemes in Indonesia. *Forest Ecology and Management* **15**:219-228.

MUETZELFELDT, R.I., ROBERSTON, B., BUNDY, A. and USCHOLD, M. 1989 The use of Prolog for improving the rigour and accessibility of ecological modelling. *Ecological Modelling* **46**: 9-34.

PINARD, M.A. and PUTZ, F.E. 1996 Retaining forest biomass by reducing logging damage. *Biotropica* **28**(3): 278-295.

PROCTER, J. 1994 Report on the soils of 15 permanent sample plots near Camp 48, Palangkaraya, Central Kalimantan, Indonesia. Internal report, Indonesia-UK Tropical Forest Management Programme, Jakarta.

RIPLEY, B.D. 1987 Spatial point pattern analysis in ecology. *In:* Legrande , P. and Legrande, L. (eds.) *Proceedings of NATO Advanced Workshop on Numerical Ecology, June 1996*. Springer-Verlag, Berlin.

SHUGART, H.H. 1984 *A theory of forest dynamics: the ecological implications of forest succession models*. Springer-Verlag, New York.

SOKAL, R.R.and ROHLF, F.J. 1995 *Biometry: the principles and practice of statistics in biological research*. Third edition. W.H. Freeman and Co., New York.

VANCLAY, J.K. 1994 *Modelling forest growth and yield: applications to mixed tropical forests*. CAB International, Wallingford, Oxon.

VANCLAY, J.K. 1995 Growth models for tropical forests: a synthesis of models and methods. *Forest Science* **41**(1): 7-42.